P2/FedX Smart Speaker Interface Spec
DS 268891

Rev 1.6

7/28/03

Revisions

Rev	Date	By	Details of Changes
1.0	2/10/03		Created initial revision, based on Extensions to the Normal Protocol document. Still need to add info about using existing Bose wall plates, etc.
1.1	2/25/03		Modified sections 3.3.1 and 3.3.2 to add a reference receiver circuit for speakers with 3.3Vrails, but no +5V rails.
1.2	3/4/03		Modified the splitter diagram in section 3.8.6 to disconnect pins 1-2 from the New Speaker end. This allows Ballpark to output audio on these pins.
1.3	5/23/03		Added Mute All Assert and De-Assert message definitions to section 9.2.3. Updated the Section on LSA.
1.4	5/29/03		Updated console message Address byte definition: now zone bits must be set to 1111 for all Polls, Queries and Set Main Attenuation messages (any spontaneous console messages unrelated to remote button presses, and which should NOT cause stream-select changes in the speaker). Added section 8.8 to describe this.
1.5	6/17/03		Refined the definition of Mute All to include ALL speakers, regardless of the source being listened-to (local sources or the two networked streams/zones). Also refined the definitions for which messages are used by speakers to control switching between networked streams. Removed the zone 1111 definition, since this is needed for Mute All assert/de-assert broadcasts. Zone switching is now limited by definition to be controlled only by On/Off messages and Pass-Key-Code messages—zone bits of all others are ignored.
1.6	7/28/03	,	Changed RoomG references to RoomO (room address for the remote controlling the variable outputs), per Marketing's . Also, integrated Bill 's re-write of Setion1 (adding Guiding Principles, etc.). Published to WindChill as DS268891 revision A (clayj).

1.0 Spec Overview

1.1 Scope

The Postman1 Normal protocol was developed to allow bidirectional control of Cobalt2-class speakers at 4800 baud. The NPP process has recently produced a need to expand the protocol to support a network of "speakers" with numerous new features. This spec seeks to define an expanded form of the Normal protocol which enables these new features. Furthermore, this spec provides the necessary hardware interface and cabling details required to properly define, create and install networked devices. To guarantee compatibility with other network devices, and to avoid corruption of networked control signals and audio streams, the guidelines described in this document should be followed as closely as possible.

1.2 Definitions

Console: A Lifestyle head unit capable of sending Smart Speaker commands.

Slave Device: A Smart Speaker or integrated system which functions as a playback device for a Lifestyle console.

Network mode: The mode in which a slave device plays back audio from the Lifestyle console.

Local mode: The mode in which a slave device plays back audio from its internal transports or a locally connected device. This audio is not passed through the Lifestyle console.

1.3 Guiding Principles

This section outlines the guiding principles behind the design of the Smart Speaker specification. This serves as a high level description of how a Lifestyle system should operate in a multi-room environment.

A primary goal behind the design of the Smart Speaker specification was to create a system with interchangeable playback devices. This allows multiple price points to be assembled by mixing and matching consoles and slave devices. In order to ensure interoperability with future devices, all slave devices must look the same to the console. They must all speak a common protocol. *The Lifestyle console should not need to know the specifics of the slave device's features or UI.* In addition to speaking a common protocol, all slave devices must have a common set of behaviors. For instance, all slave devices must respond to mute, unmute, and volume commands in a similar manner.

Another goal behind the design of the Smart Speaker specification is to provide the user with a seamless experience across multiple rooms and multiple remotes. The system should behave largely the same in a remote room as it does in the main room. A slave device should function largely the same when using the Lifestyle remote as it does with its native remote. In order to accomplish this seamless behavior, the Lifestyle console must dispatch commands received from an RF remote to the appropriate slave device. To ensure interoperability with future devices with new features, the Lifestyle console must be able to pass ***unknown*** commands through to the slave device. *The console need not know what the command is.* In this respect the console functions as a router, determining where a command came from and where it should be sent to. In some cases (e.g. a source change) the command is sent to the console's command processor. In other cases (e.g. an unknown command), the command is routed directly to the slave device. This decision is based only on the state of the Smart Speaker and the command received. In this way it is possible to create a system that is extensible to new types of slave devices.

Some slave devices will themselves be systems, and will include control integration for other connected devices. Such control automation will normally be accessed through the slave device's native remote. It is logical to ask how that control automation could be accessed from the Lifestyle remote. Such control automation is available to the extent that it can be supported via the above routing mechanism. *No additional processing will be done by the Lifestyle console to accommodate control integration at the slave device.*

1.4 New Features

• Some speakers may be hardwired to the network, and others connected through Arrow-class wireless transceivers. Such transceivers will transfer 1-way audio from the console to networked speakers, and 2-way Smart Speaker control information between the console and speakers. Latencies for such transceivers need to be supported.

• Speakers will no longer be dedicated (necessarily) to a specific zone. Instead, the 15 networked speakers will be able to dynamically jump from one zone to another. Changes may need to be made in the protocol's addressing to allow this.

• Speakers may have a local means for choosing between Postman2/FedX networked audio or its own local sources (Ballpark will have an internal tuner and CD player, for example). Such means may include source select buttons on the speaker itself, or on local remote controls (IR, for example). The protocol must provide a method for identifying when speakers are and are not listening to networked audio, so the console can power up/down appropriately. A polling structure needs to be implemented to identify these changes.

• Variants of RF remote controls should be capable of controlling both the Postman2/FedX console as well as the speakers' local sources. The protocol needs to expand to allow transport, volume/mute and other proprietary speaker-only keys from these remotes to be relayed down the Smart Speaker bus for use by the speaker at any time, including in local playback mode or OFF mode.

• **Lower priority**: Local buttons on the speakers themselves may need to control the FedX console. The protocol needs to support a means for transmitting these key commands from the speaker back to the console. Potentially, this should also include a means to make more direct changes to FedX system status variables (source being played, track, station, presets, etc.).

• **Lower priority**: Speakers may have local VFD's, etc., capable of displaying FedX source information as well as local source information. The protocol needs to define a means for broadcasting FedX source-related information to interested speakers, as well as (potentially) a means for specific speakers to request such information on demand.

• **Lower priority:** Ballpark-class products may need to push display information onto a local Hermes-class remote control (when Hermes is controlling Ballpark's local sources). The protocol needs to expand to allow source-related information to be passed up to the console, to be brokered out to the RF remote.

1.5 Operational Overview

This section contains a brief overview of how a multi-room Lifestyle system operates under normal circumstances. It is intended to give the reader a basic understanding of what goes on in the system, without all of the detail. It is not intended to cover every possible scenario, and is not a substitute for the complete documentation found later in this document and others.

Under normal operation, the Lifestyle head unit continuously polls all connected Smart Speakers to determine if they are on the network and turned on. There are four possible states for a Smart Speaker: **"On"**, **"Off"**, **"Local"** or **"Not Responding"**. The console's command processing proceeds according to the state of the speaker as follows:

In all states the Lifestyle head unit will route volume and mute commands from the Lifestyle remote to the slave device (with the matching room code) via the Smart Speaker protocol. When, as a result of polling, the Lifestyle console discovers that a slave has been turned "on," it will light up the appropriate zone and begin playing the last selected source. If the zone is already playing, the speaker will join the current source. In the event that the Lifestyle console determines that all speakers connected to a given zone are either "off" or "local," it will power down that zone.

When the slave device is in the "On" state, transport controls are processed natively by the Lifestyle's internal transports or by the UEI subsystem. Keycodes that are not native to the Lifestyle unit will be passed through to the remote device. This allows the Lifestyle console to be extensible to new classes of slave devices which may require additional controls beyond those currently envisioned.

When the remote device is in the "Local" state, all controls, including transport controls, will be passed through the Lifestyle head unit to the remote device. Transport controls will **not** be processed locally by the Lifestyle head unit. This allows fully context sensitive transport controls across all the Lifestyle internal transports as well as the remote device's local transports.

When the remote device is in the "Off" state, all controls, including transport controls, will be passed through the Lifestyle head unit to the remote device. This allows the remote device to be powered up by means of sending a local source select command or an On/Off command. In the case of an "On/Off" command, the remote device will power up in its previously selected source. If the source was the Lifestyle head unit, the state is set to "On," which will be caught in the next polling cycle. This scheme is highly scaleable in that each remote device manages its own behavior, and the Lifestyle needs to know only a minimal amount about each remote device.

When the remote device is in the "Not Responding" state (as will be the case for third-party amplifiers, for example), all controls, including transport controls, will be passed through the Lifestyle head unit to the remote device, exactly as for the "Off" state, above. However, the console will manage on/off state variables for these slaves by keeping track of on/off and source select keypresses from their remotes. This is necessary so that the console will know when to properly shut-down a zone (when all rooms using it are off). Note: if a slave powers-up and begins replying on the network, the on/off state information in its replies will be given higher priority, and used to update the console's on/off state variables for that room.

2.0 Features of the Postman2/FedX Smart Speaker Bus

2.1 Differences from Postman1

The Postman2/FedX Smart Speaker protocol is based on the Normal protocol used in Postman1, but includes some significant changes. The nature of these changes **makes the two protocols incompatible**. In brief, they include:

- Increasing the signaling speed from 4800 bps (bits per second) to 19.2k bps.
- Tighter restrictions on message packetization and message/reply timing.
- Adding a direction bit to message headers (indicating whether a message is being sent by the P2/FedX console, or by a speaker).
- Changes to the speakers' Address byte.
- Eliminating the 1-byte speaker ACK. Speakers now reply with a 4-byte Poll_Reply message as a default, but can substitute higher-priority messages, if any are pending.
- Implementing a new polling cycle, by which the console remains in continuous contact with all speakers.
- New message type definitions, including a means for passing-through remote control keypresses from the console to speakers, and vise-versa.
- A new capability for speakers to reply with all message types.

Details of each of these changes are provided in the next sections.

2.1 Number of Speakers Supported per Zone

Postman1's Normal protocol allowed up to 7 speakers on each of its two sub-networks (Zone1 and Zone2), each with addresses from A to G. To fully identify a given speaker, the system needed both room and zone addressing. This has changed somewhat for P2/FedX.

The P2/FedX Smart Speaker protocol is defined to allow a console to support **up to 15,** total, networked speakers, each with a unique address (no more than one speaker may use a given address). These addresses are referred to in this document as RoomA, Room B, RoomC, etc., up to RoomO. A 15th address (RoomP, effectively) is reserved for broadcast messages (intended for ALL speakers). **These 15 speakers may play EITHER the Zone1 or Zone2 audio signals (or neither) at any given time,** in response to remote control commands. Therefore, for P2/FedX, the room address ALONE identifies a specific speaker-- any zone information sent to a speaker simply informs it of the audio stream meant to be played.

2.2 Plug and Play Capability

New speakers can be added to the network at any time, without powering-down either the network or the speakers, and without turning off the console.

2.3 Support for LSA1 on Zone2

Because the Postman2/FedX console has two separate (though time-shared) Smart Speaker buses, the console can use two different communication protocols for each Speaker Output connector. Cobalt2 will be connected to the Zone1 Speaker Output (using the protocol in this document). If no other new speakers are required to be attached to the Zone2 Speaker Output, its protocol can be set via the OSD to “Legacy”, to support connecting a single LSA1.

2.4 Support for a Single Legacy Variable Speaker

One speaker address will be shared with the P2/FedX Zone2 variable audio outputs. Volume Up/Down commands from an RF remote control set to this address with initiate Smart Speaker messages as usual, but will ALSO control the internal volume control chip feeding this Zone2 variable output (available from the Zone2 Speaker Output mini-DIN on the back of the console).

Presently, this shared address is defined to be RoomO. Note: Postman1 used RoomG.

3.0 Physical (Hardware) Interface

The Postman2/FedX Smart Speaker Interface uses essentially the same circuit as Postman1. The physical interface remains the same, with slight improvements to tolerate the increase in bit rate and increased speaker loading. The complete hardware interface consists of the bus (control data) interface, audio interface, and a +10V Turn-On signal to support legacy speaker systems. **NOTE: no attached network device may disable network communication or degrade its audio quality at any time, including while OFF or while unplugged/unpowered.**

3.1 Overview of the Physical Bus

The Smart Speaker Bus is a two-wire (Data and GND), bidirectional, half-duplex interface intended to be connected from the Postman2/FedX console to each networked Smart Speaker. All data and control messages are sent via this bus. Messaging is fixed at 19.2kbps, and follows packetization and timing rules defined in this document.

Network lengths up to 150 feet (from the console to the furthest speaker on any stub) can be supported, with cabling following either a daisy-chain or star configuration (or combinations thereof)-- the only requirement is that electrically all networked speakers must be wired in parallel. Provisions have also been made to support use of the relatively high capacitance, yellow, 8-conductor, Bose-provided speaker wire sold through PTS. Suggested connection guidelines are provided below.

Loading has been calculated to support up to 15 networked speakers, provided that they follow the hardware guidelines defined here.

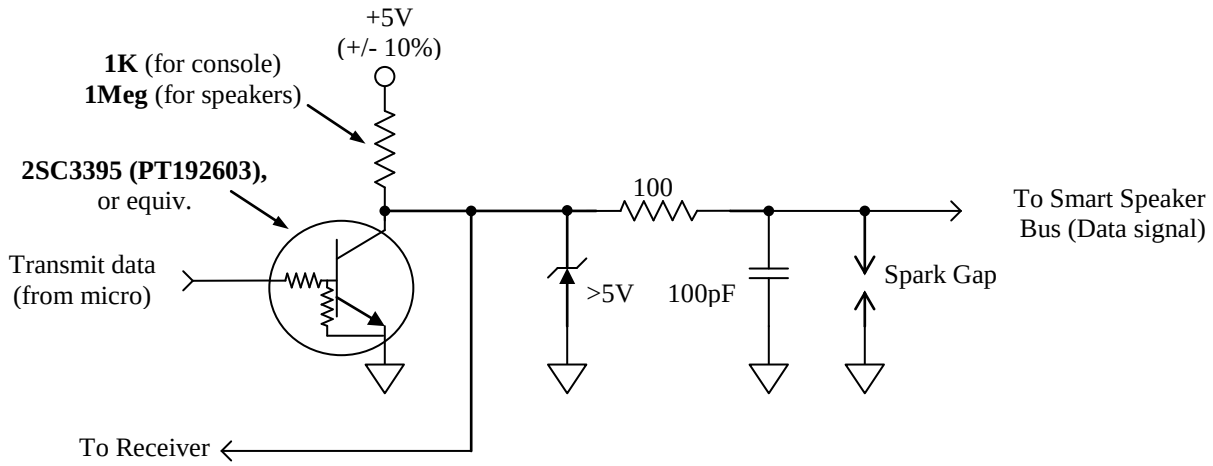
The Data signal has a 1K Ohm pullup resistor to +5V at the console end, and idles at +5V (normally high) when no messages are being sent. Networked speakers may optionally include a 1Meg Ohm pullup resistor to a local +5V source to bias the Data signal to a known state when disconnected from the console. All signaling (from the console as well as networked speakers) is achieved through open-collector transistors which pull the Data signal to a local GND. More detailed circuit requirements are provided below.

3.2 Transmitter Details

As described, each device interfacing to the bus must signal through an open-collector transistor capable of tolerating +5V, and able to fully pull-down (guaranteeing saturation) with the worst-case pullup resistance (about 650 Ohms, factoring in the effect of 15 receivers). Since the bus will be strung throughout a home, and potentially past significant electrical noise sources, filter/protection components should also be provided **(100pF, max on bus).**

3.2.1 Reference Transmitter Circuit

For reference, the suggested output section of a bus transmitter is:



NOTE: if the micro uses a standard UART, another logical inversion may be needed, preceding this stage.

3.2.2 General Transmitter Electrical Requirements

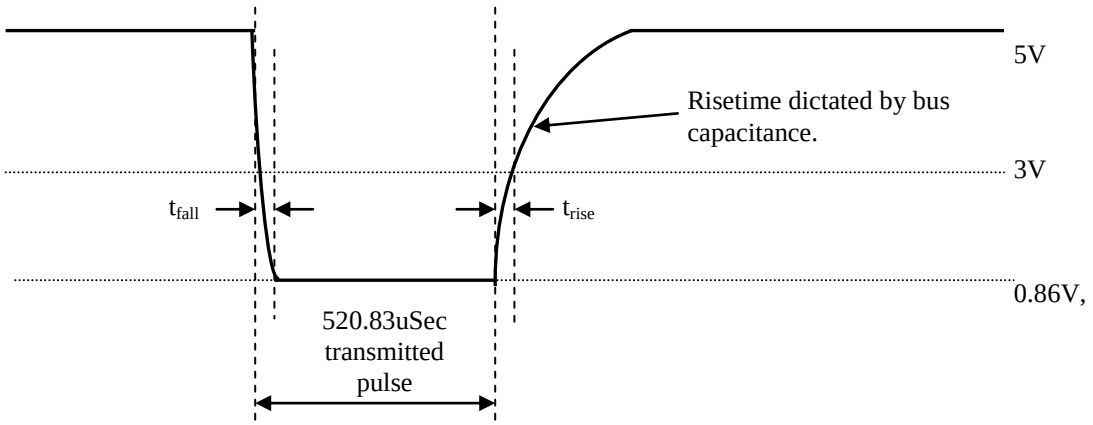
Regardless of the transmitter topology used, it is essential that its parameters be controlled as follows:

Parameter	Allowable Value
Capacitance connected directly across the bus (Speaker Data to ground)	105pF, max (100pF, 5% capacitor, max).
Transmit logic Low **on the bus**	< 1.0V, max. with a test circuit consisting of a 618 Ohm pullup resistor from Speaker Data to +5V (modeling the effect of 15 worst-case receivers). This is roughly equivalent to the performance of a saturated NPN with 100 Ohms series resistance, as suggested.
Transmit logic High **on the bus**	High impedance at the transmitter (open-collector NPN). Local transmitters may have no less than 1M Ohm, min, pullup to a local +5V rail.

3.2.3 Worst-Case Drive Waveforms

120 feet of Bose speaker cable would load the network with about .01uF of extra capacitance. Therefore, the following would represent a worst-case drive waveform for a start bit driven by a slave onto a maximum-length network loaded by the worst-case number of speakers. Drawing is NOT to scale. Note the widening of the bit due to bus capacitance. Bit falltimes should be about 10 times faster than risetimes under all situations.

Maximum-length network (worst-case) drive waveform using suggested transmit and receive circuits:



Parameter	Worst-Case Value
Maximum expected total load capacitance	.012uF, including cable capacitance and 105pF filter capacitors per network device.
Worst-case data fall time (t_{fall})	**0.6uSec** (time to fall from 5V to 3V, with .012uF capacitance and a maximum number of network devices).
Worst-case data rise time (t_{rise})	**11.2uSec** (time to rise from 0.2V to 3V, with .012uF capacitance and minimum number of network devices).

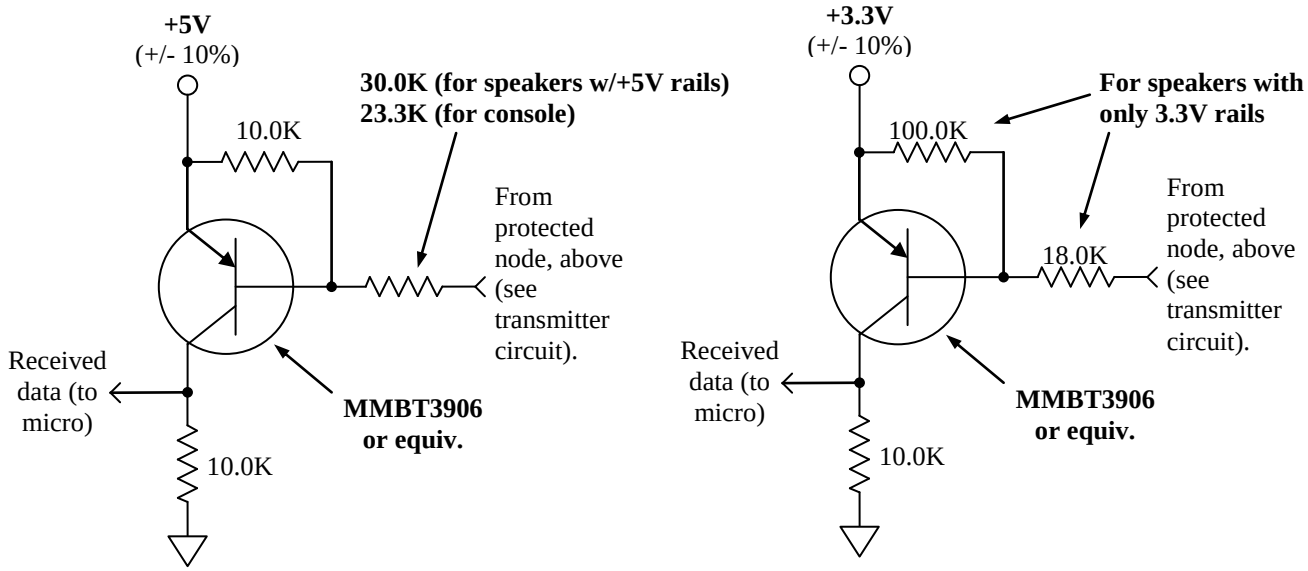
3.3 Receiver Details

Bus receivers must detect low-going transitions on the network and relay them to the micro. The following circuit sets a low-going receive threshold at about 2.6V for speakers and 3V for the console. No hysteresis is provided in this example-- therefore some multiple-transitions may be experienced on the edges of noisy bits. Devices must add hysteresis in the event that their receive algorithms cannot tolerate these transitions. Standard UARTs and other schemes which sample bits near the center of bit cells, for example, will optimally tolerate multiple transitions. Receive schemes which decode data by using edge-triggered interrupts to measure network high/low times might require hysteresis, unless sufficient delay exists in the interrupt service routine before re-arming. Ideally, software should implement a debounce scheme to implement pseudo-hysteresis, regardless of the circuit used.

Bus receivers must also tolerate network transients. The receiver circuit is therefore tapped off the protected node of the transmitter output stage, as shown in the Transmitter Details diagram above. NOTE: in this topology, all data transmitted by a network device will be fed back into its own receiver. Software must be robust enough to tolerate this.

3.3.1 Reference Receiver Circuits

For reference, the suggested +5V and +3.3V bus receiver circuits are:



NOTE: if the micro uses a standard UART, another logical inversion may be needed, following this stage.

3.3.2 General Receiver Electrical Requirements

Regardless of the receiver topology used, it is essential that its parameters be controlled as follows:

Parameter	Allowable Value
Low-Going Receive Threshold	2.6V, nominal, for speakers. 3.0V, nominal, for console.
Maximum allowable current drawn from the bus when in the logic "high" state.	33uA, max. (Equivalent to a 150K Ohm resistor to ground.)
Maximum source current drawn from a speaker receiver when the bus is in a logic "low" state.	150uA, max. (Equivalent to a 33.3K Ohm resistor to +5V, or a 22.0K Ohm resistor to +3.3V.)

3.4 +10V Turn-On Line

To support legacy AM5P/AM20P and 901P speakers, the Postman2/FedX console will provide a +10V Turn-On signal, with a drive circuit identical to CD-20, LS50 and LS28/35 (Postman1). This signal will be low (2.2K pulldown to GND) when no speakers are active on a given Zone, and high (current-limited high-side PNP switch, providing from +8.8V to +10V depending on load). Maximum drive capability is about 75mA at +8.8V. Therefore, if speakers use this line for any other purpose (Energy Star, etc.), care should be taken to avoid drawing more than 5mA per speaker (5mA = 75mA / 15 speakers). Any such current draw should also be only temporary, to reduce console power dissipation.

3.5 Audio Interface Details

The complete Smart Speaker interface consists of audio signals as well as the control bus just described. The audio signals for all new networked devices are fixed output (always sent at full volume)-- requiring any volume control to be implemented in the speakers. This guarantees maximum signal to noise at the speakers. A variable output is also provided, however, to support legacy speakers. To support the Cobalt2 speaker system in the main room, a digital output is provided.

3.5.1 Console S/PDIF Output for the Main Room

For the Main Room speaker system (Cobalt2 for the Postman2/FedX system), the console provides a S/PDIF-based differential digital audio stream on the Zone1 Speaker Output mini-DIN, pins 1 and 2 (see connector pinout, below). This stream is configured for a (non-standard) 3Vpp output level, when loaded by the speaker, and is transformer-isolated and balanced. 390pF filter caps to ground are added to reduce emissions.

This stream will be limited to a 48kFPS (48K frames/Sec) data rate, maximum, when connected to Cobalt2 (which cannot handle streams with faster rates). The Postman2/FedX console hardware is capable of generating up to 192kFPS streams, however, which may be put to use in future variants of the product, with future speaker systems. The Postman2/FedX console will output digital audio in PCM, AC-3, DTS, MPEG2 and possibly AAC formats. The compressed formats must be identified and decoded by the Cobalt2 bass box.

Only one speaker is intended to connect to this S/PDIF audio stream.

3.5.2 Console Analog Outputs for Non-Main Rooms

To connect to non-main room speaker systems throughout the house, a pair of analog stereo outputs (Zone1 and Zone2) are provided on the Zone2 Speaker Output mini-DIN connector (see pinout, below). As mentioned, these left/right pairs are both fixed-output. Full-scale (maximum) output signals are about 2Vrms, with typical playback levels being about 300-400mVrms. Playback levels for all console internal and external audio sources have been gain-scaled in the console to play at equal amplitudes.

These outputs are standard, single-ended analog outputs, driven essentially by op amps, with about 50 Ohms added series resistance on each output. 47uF DC blocking capacitors are added in the console to each output signal, followed by 100K resistors referencing each to the console's analog ground. It is expected that this output impedance can drive up to 15 speakers (with input stages defined below) without suffering either unwanted attenuation or loss of low-frequency response. Cabling guidelines are provided below to guarantee proper zone/zone isolation and noise shielding.

The Zone2 Speaker Output connector pins 1 and 2 also provide a left/right variable analog signal pair, for variable-input legacy speakers (AM5P/20P and 901P). These signals are identical to the Zone2 fixed outputs on pins 3 and 4, but are able to be attenuated by the Zone2 volume control chip inside the console.

3.5.3 Differential Inputs for Networked Speakers

Long lengths of audio cable routed through a house have been found to be susceptible to picking up audible amounts of noise. Speakers interfacing to the Smart Speaker bus should therefore configure differential amplifiers at their audio input stages. As a reference for these diff amp inputs, a dedicated Audio Reference will be sent from the

console and included with the other network conductors. See cabling details, below. The full list of requirements on a networked speaker's audio interface are:

• Zone1 and Zone2 input diff amp circuits should be identical (same gain, noise floor and bandwidth, consistent with the audio quality goals of the speaker).

• Diff amps should be used, **with all legs equally balanced**.

• **Resistance looking into each leg from the network should be 20K Ohms or greater**. This allows diff amps configured with 10K Ohm resistors in each leg, for example.

• Each leg should be capacitively coupled to the network, with all capacitors of equal value. Each networked device is free to choose their own capacitor values, based on desired frequency response.

• The dedicated Audio Reference signal should be used for all diff amps: Zone1 Left/Right as well as Zone2 Left/Right. To avoid audio currents flowing back into this signal which could destroy Zone1/Zone2 isolation, **Audio Reference should be used for the NON-INVERTING legs of the diff amps only.** In general, no more than 1 microAmp, rms, of audio current should be induced onto the Audio Reference signal. This maintains 92dB isolation (considered an acceptable minimum).

3.5.4 Zone1/Zone2 Input Selection MUX for Networked Speakers

All Smart Speakers developed to interface with Postman2/FedX will have the ability to select one of two possible audio streams: the Zone1 stream or the Zone2 stream. Selection will be controlled via Smart Speaker commands. Speakers should therefore provide a 2-input, stereo audio MUX at its network audio input, under control of the micro managing Smart Speaker messages. NOTE: configuring this MUX ahead of the diff amps, to save cost, must be done carefully, to avoid diminishing their performance or violating the above requirements.

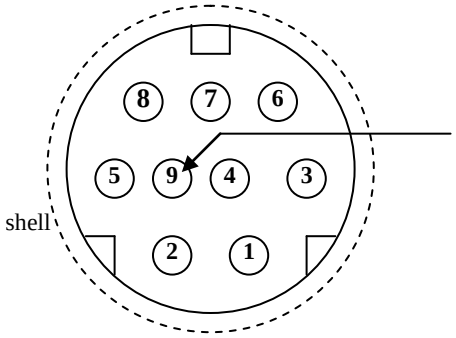
3.6 The P2/FedX Console Speaker Output Mini-DIN Connectors

The Postman2/FedX console has two 9-pin Speaker Output mini-DIN connectors, one for Zone1 and one for Zone2. Their pinout is as follows:

ZONE1:
Pin 1: S/PDIF0 digital audio signal for Cobalt2 (Z1_NET0).
Pin 2: S/PDIF1 digital audio signal for Cobalt2 (Z1_NET1).
Pin 3: Zone1 fixed Left analog audio signal (Z1_LEFT). CANNOT be made variable.
Pin 4: Zone1 fixed Right analog audio signal (Z1_RIGHT). CANNOT be made variable.
Pin 5: GND.
Pin 6: +10V Turn On signal for Zone1 (Z1_TURNON).
Pin 7: Smart Speaker Data for Zone1 (Z1SPKR_DATA).
Pin 8: GND.
Pin 9: No connect.
Shell: GND.

ZONE2:
Pin 1: Zone2 variable Left analog audio signal (OUTLVAR).
Pin 2: Zone2 variable Right analog audio signal (OUTRVAR).
Pin 3: Zone2 fixed Left analog audio signal (Z2_LEFT).
Pin 4: Zone2 fixed Right analog audio signal (Z2_RIGHT).
Pin 5: Buffered Zone1 fixed Right analog audio signal (BZ1_R). Will be shorted by old cables.
Pin 6: +10V Turn On signal for Zone2 (Z2_TURNON).
Pin 7: Smart Speaker Data for Zone2 (Z2SPKR_DATA).
Pin 8: GND.
Pin 9: Buffered Zone1 fixed Left analog audio signal (BZ1_L).
Shell: GND.

Looking into a 9-pin mini-DIN connector (as when plugging into it) the connector pins are:



3.7 The Input Mini-DIN Connector for Networked Speakers

Networked Smart Speakers will likewise have a 9-pin mini-DIN connector for connecting to the Smart Speaker bus. This one connector must take in Left/Right audio for both Zone1 and Zone2, a well as the Data line, digital ground, and a separate ground to use as a reference for the audio diff amps. Optionally, the +10V Turn On signal could be brought in, as well. Although all-together this only amounts to only 8 conductors, the speakers will be defined to use a 9-pin mini-DIN identical to the console's (though only a single, as opposed to a dual) to allow cables to be reversible.

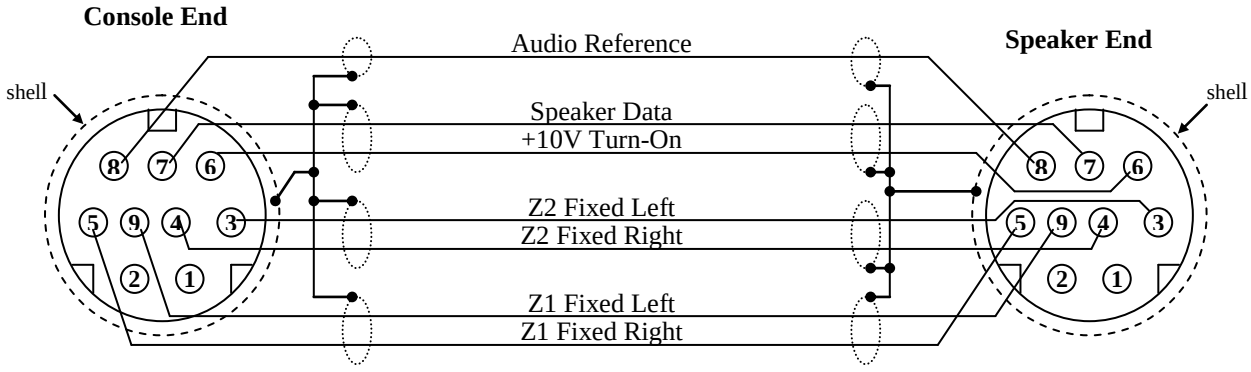
The speakers' mini-DIN will be pinned-out as follows:

Pin 1: Unconnected (console's Zone2 variable Left analog audio).
Pin 2: Unconnected (console's Zone2 variable Right analog audio).
Pin 3: Zone2 fixed Left analog audio signal.
Pin 4: Zone2 fixed Right analog audio signal.
Pin 5: Zone1 fixed Right analog audio signal.
Pin 6: +10V Turn On signal for Zone2.
Pin 7: Smart Speaker Data for Zone2.
Pin 8: Audio Reference (dedicated ground for diff amps). NOT tied to speaker's product ground.
Pin 9: Zone1 fixed Left analog audio signal.
Shell: GND, used as the speakers' digital (product) ground. NOT shorted to pin 8 GND in the speaker.

3.8 Cabling Details

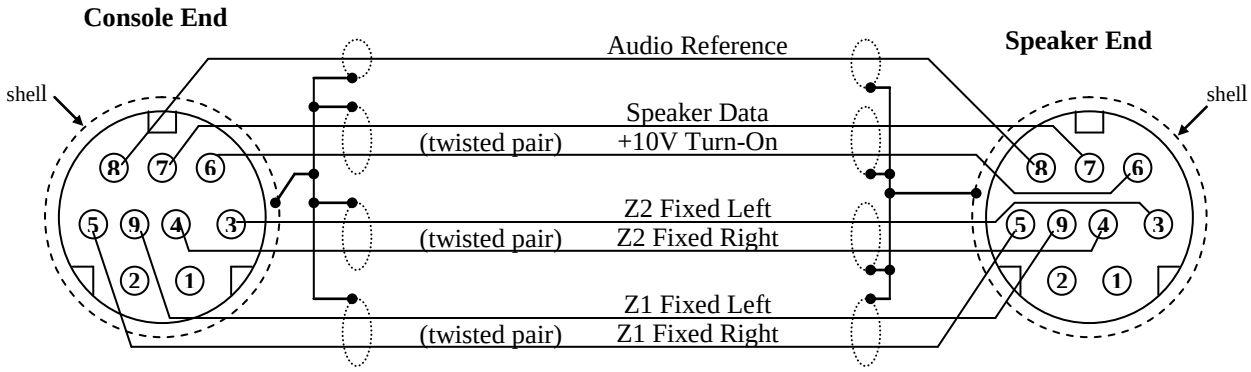
3.8.1 Bose Pre-Terminated Cables ("Zone 2 Expansion Cables") for New Speakers

Cables provided by Bose for customers to connect new-style Smart Speakers (A2, LSA2, Ballpark, etc.) to the P2/FedX console Zone2 Speaker Output will be constructed as a 1:1 pass-through of pins 3 through 9, with separate shields as follows. Note: pins 1 and 2 are not connected, and the cable is **reversible.**



3.8.2 Using the Bose System Cable (257187) with New Speakers

Professional installers, when using the 8-conductor Bose System cable, part number 257187 (4 shielded, twisted pair) to connect new-style speakers to the console, should make connections as follows (identical to above, using twisted pair):



3.8.3 Connecting Legacy AM5P/20P Speakers

AM5P/AM20P's CANNOT plug into the Zone1 Speaker Output of P2/FedX. The Zone1 connector left/right audio outputs cannot be configured as variable outputs (as Postman1's could, via the OSD), so these speakers are not supported in Zone1.

AM5P/AM20P's can plug directly into the Zone2 Speaker connector and operate properly, and can co-exist with newer speakers (LSA2, A2 and Ballpark). They use the Zone2 Variable Left/Right signals on pins 1&2, as well as the +10V Turn-On Line on pin 6. Pin 8 is used to shield the Turn-On line and provide speaker ground. The shell ground is used for Audio Reference in this speaker. The same cables used to connect these speakers to CD20 (LS20) and the MRI (LS40) can be used here. **NOTE: these cables will internally short pin 5 to the shell ground, making the Zone1 Fixed Right output unusable for new speakers-- a special splitter must be used to prevent this (connecting the console shell to the splitter output's pin 5, but leaving this output's shell floating) if these speakers need to share Zone2 with new speakers.**

3.8.4 Connecting Legacy LSA1 Speakers

LSA1's CANNOT plug into the Zone1 Speaker Output of P2/FedX. The Speaker Data signal on the Zone1 connector is used for communicating to Cobalt2, and must therefore use the Normal protocol. The LSA1 used the Legacy protocol, which cannot be supported on the same bus as the Normal protocol.

A single legacy LSA1 CAN plug into the Zone2 Speaker Output of P2/FedX, with the same cable used to plug LSA1 into CD20 and MRI (LS40/50). This cable picks-up the Zone2 Fixed Left/Right signals on pins 3&4, as well as the Speaker Data signal on pin 7. Pin 8 ground is picked-up for product ground and the shell ground is used to shield audio conductors and as a reference for the LSA1's diff amps. **NOTE: The LSA1 requires the P2/FedX console to be set to the Legacy protocol via the OSD, which will make it unable to simultaneously support any new-style speakers (LSA2, A2, Ballpark, etc.).**

3.8.5 Connecting Legacy 901P Speakers

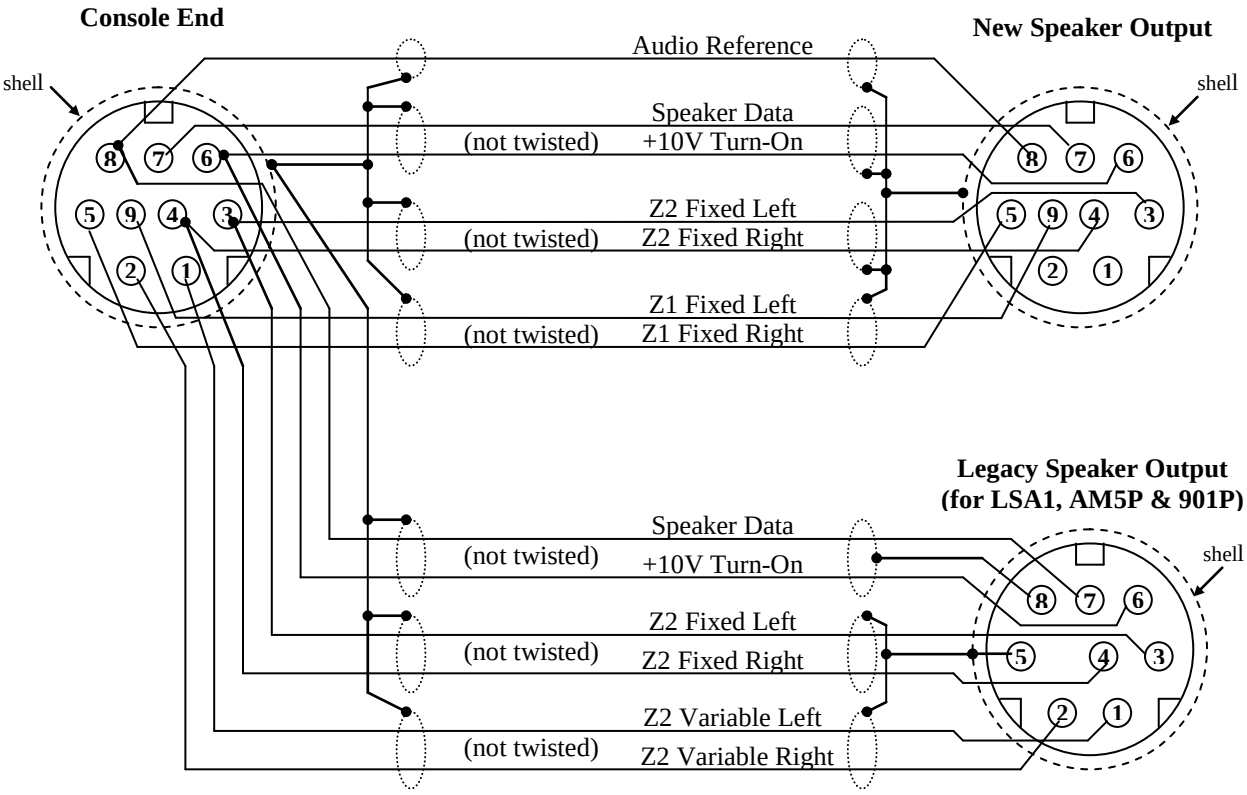
901P is another variable-input speaker, like the AM5P/AM20P. Therefore, it CANNOT be plugged into the Zone1 Speaker Output (which has only fixed outputs-- see AM5P/AM20P, above).

901P CAN co-exist on Zone2 with newer speakers. It uses the same signals as AM5P/AM20P, in a similar way. **NOTE: its standard CD20/MRI cable also shorts pin 5 to the shell. As above, it will therefore also require a special splitter if newer speakers are to share the Zone2 signals.**

3.8.6 Mixing Legacy and New Speakers On the Same Network

As mentioned above, if any of the existing Bose speaker cables are plugged into the Zone2 Speaker Output of Postman2/FedX, the Zone1 Fixed Right audio output signal on pin5 will be shorted to ground. This is a result of the fact that older consoles used pin5 as a Room Sense input (consoles would monitor this pin, and know that a cable was plugged-in when they saw this pin shorted to ground). Since the Fixed Right audio output is buffered, it is safe to short this signal to ground **if newer-style speakers do not need to be supported**.

However, if new speakers need to be mixed with legacy speakers, a special splitter must be used. The connections in this splitter would be as follows (Console end is male, Speaker Output ends are female):



The splitter's Legacy output has the following features:
1. Cables plugging into it which short pin5 to the shell will not short Z1 Fixed Right to GND.
2. Its pin 8 ground can be used for digital GND (for the Data signal, as well as +10V Turn-On).
3. Its shell ground is shorted to pin5, and can be used for Left/Right audio shielding.

The splitter's New Speaker output is a full pass-through of pins 3 through 9 (and shell) from the console mini-DIN. New Speakers will never use the pin1-2 Variable audio signals, and in fact may use pins 1-2 as audio outputs for accessories (Ballpark pedestals, for example), so these are not included in the splitter.

3.8.7 Using Existing Bose Wall Plates, etc.

This needs to be fleshed-out.

4.0 Low-Level Protocol Basics

In general, the Smart Speaker protocol remains a one-wire (two wires, including ground), bidirectional, half-duplex, signaling scheme between the P2/FedX console and a set of networked speakers.

All communication is sent via the Speaker Data conductor on the cables run from the Speaker Output mini-DIN connectors on the back of the P2/FedX console to the networked speakers. The idle state for this signal is a logical high (+5V), pulled-up via a 1K Ohm resistor in the console (see the previous Physical Interface section). All transmissions are achieved by directly keying open-collector transistor stages in the console and speakers.

The P2/FedX Smart Speaker protocol is **modeled after RS-232, with a 19.2k bps bit rate, one start bit, one stop bit and no parity bits** (identical to Postman1, except that the bit rate has been increased from 4800 bps to 19.2kbps). Message bytes are defined later in this document. **NOTE: All message bytes are sent LSB (least significant bit) first, per the RS-232 standard.**

4.1 Allowable Logic Levels

Logic levels for signaling are as follows (as seen on the Speaker Data conductor of the Smart Speaker bus):

Message Bit	Logic Level on the Smart Speaker Bus	Notes
Idle State	Logic High	Console pullup to +5V.
Start Bit	Logic Low	Open-collector pulldown to GND.
Data 0	Logic Low	Open-collector pulldown to GND.
Data 1	Logic High	Console pullup to +5V.
Stop Bit	Logic High	Console pullup to +5V. NOTE: same as return to Idle state.
Xmit Logic Low	1V, max on bus.	Only one device transmitting. 1K pullup to +5V at the console.
Xmit Logic High	4.5V, min on bus.	With 15 receiver loads. 1K pullup to +5V at the console.
Rcv Logic Low (slaves)	< 2.6V	Console pulls all the way to .1V.
Rcv Logic Low (console)	< 3.0V	Slaves pull down through series resistors.
Rcv Logic High (slaves)	> 2.6V	
Rcv Logic High (console)	> 3.0V	

4.2 Allowable Bit Widths

The 19.2kbps data rate used by all networked devices must be guaranteed to be within 0.5% of the nominal rate. This ensures edge accuracy out to each byte's Stop bit of +/- 5% (+/- 10%, assuming worst-case timing error for both the transmitter and receiver). Allowable bit widths are therefore defined to be as follows:

Message Bits	Bit Widths	Tolerance
All (Start Bit, Stop Bit, Data 0, Data 1)	**52.083uSec**, nominal.	+/- .5% (+/- .26uSec). NOTE: Allowable error includes all clock error due to crystal accuracy, temperature drift and aging, etc.

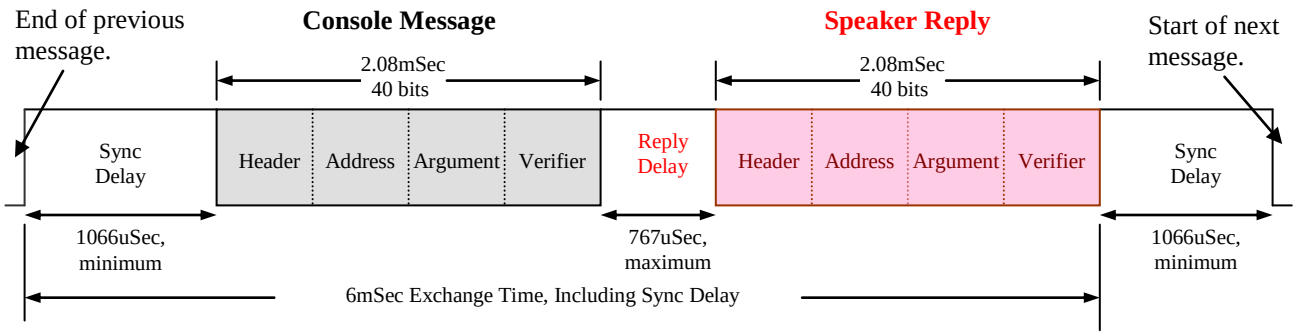
NOTE: these tolerances apply to bit widths as generated by bus transmitters. Actual received bit widths may be distorted on worst-case networks (see Section 2.2.3).

4.3 Message Timing Requirements

To guarantee speaker synchronization, messages will be formally required to be packetized (sent as one continuous burst, without delays between bytes), with inter-message timings strictly controlled, as follows:

Parameter	Requirement	Notes
Longest logic High time within the body of a message.	*9 bits = **469uSec,** nominal, at 19.2kbps.*	*This would correspond to a byte with value 00h. Included here for reference only.*
Allowable delay between message bytes (console or speakers).	0 uSec.	All message bytes must be sent out as one continuous burst, without pauses between.
Required network idle time before sending console messages ("Synchronization Delay")	1.066mSec, min.	End of the last stop bit (from speaker or console) to beginning of console's start bit.
Allowable delay between the end of a console message and the start of a speaker's reply.	767uSec, max.	End of the console's last stop bit to the beginning of the speaker's first start bit. **True for ALL types of replies.**
Allowable delay between console queries and speaker replies.	TBD by high-level software.	Speaker reply, when ready, must wait to be sent immediately following (within 767uSec) the next poll from the console.

Message example for a common 4-byte command and reply, as seen on the network (note: bus idles high):



4.4 Guaranteeing Message Synchronization

As shown in the table above, the console always needs to ensure that the network has been idle for at least 1.066mSec before initiating a new transmission. This delay is intentionally longer than the allowable delay between console messages and speaker replies, so that devices on the network can use simple timeouts to re-synchronize with the console whenever necessary (ideally, on an ongoing basis). The recommended timeout for speakers to use for re-synchronizing to the console is **916**uSec (916uSec = 767uSec + (1.066mSec - 767uSec)/2.). **In other words, if a speaker identifies that the network has been idle for 916uSec, that speaker can re-arm its receive routine and will properly capture the next console message from its beginning.**

Speakers can identify the end of an inbound console message by waiting for the first network idle time of safely greater than 469uSec (the longest possible string of 1's within the body of a message). The recommended timeout is about 521uSec. This gives speakers 246uSec to examine the received message and begin its ACK/reply, as required, before 767uSec.

If these message timings are adhered to, no loss of synchronization should occur (network devices should never improperly lose track of where the beginning and end of messages are). Nonetheless, message address and verifier bytes should be examined to confirm that all received messages are valid. Furthermore, if a network idle time long enough to identify the end of a message is detected, any partially-received messages less than 4 bytes long should be deleted, and re-synchronization should take place.

4.5 Recommendations for Software Drivers

4.5.1 Optimized Low-level Receiver

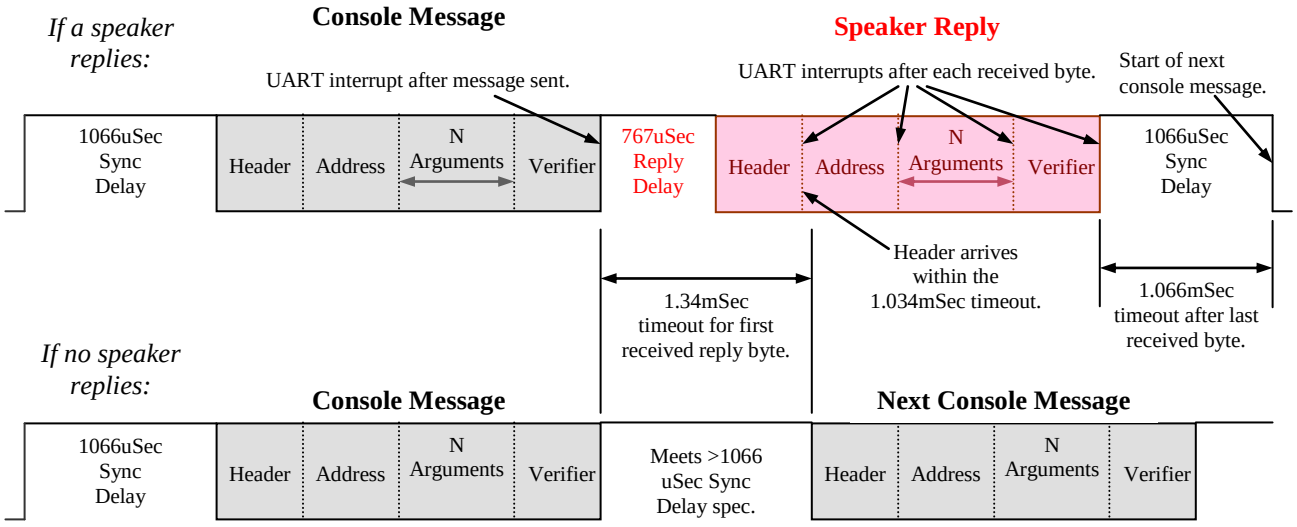
Standard hardware UART receivers provide optimum protection against noise and jitter/slew distortion, and are recommended. To approximate this performance in software where hardware UART's are not available, a bit-banged receive routine should work as follows:

1. After synchronization has been confirmed and an inbound message can be expected, the data input should be armed with an edge-triggered interrupt.
2. When the first edge of a message is detected (the first start bit of the first byte), it should be debounced by immediately re-sampling 1-2 more times to reject noise (though noise should be extremely rare on typical networks). If found to be valid, a state machine driven by auto-reloaded 52.083uSec timeouts should be implemented to sample the next 10 bits as close to the center of the bit cells as possible.
3. The first 8 stages of this state machine will collect message bits. Each of the 8 message bits should be quickly sampled 1-3 times at the center of the bit cell, and the proper value stored away.
4. The timer interrupt routine for the 9th bit (the stop bit) should confirm that the bus is in the idle (high) state, and re-arm the edge-triggered interrupt for the next byte's start bit.
5. If a new byte's start bit edge is detected before the 10th 52.083uSec interrupt, cancel the 10th timer interrupt and assemble a new incoming byte as before (using this edge to re-synchronize the timing of the 10-bit sampling engine). However, if the 10th 52.083uSec interrupt expires first, the message is over. Process accordingly. Slaves can immediately begin generating their reply.

4.5.2 Transmit/Receive Timing for the Master

The Postman2/FedX console will have a hardware UART assigned to the Smart Speaker interface. The following scheme will minimize the interrupts associated with managing this UART:

1. Transmit the full outbound message at once (N bytes), setting the UART to interrupt the system when done. For messages longer than the16 byte maximum UART buffer in the CS98200 (rare), interrupts will need to be used to transmit the messages piecemeal, in the largest blocks possible (up to 16 bytes). Note: this should be done as to introduce no noticeable delays between bytes.
2. Once the outbound message is finished, enable the UART receiver and set it to interrupt after one received byte (first interrupt should occur within 1.288mSec where 1.288mSec = 767uSec Reply Delay + 520.83uSec first byte length). Also arm a timer interrupt to expire in about 1.34mSec.
3. If the 1.34mSec timer interrupt expires before the first full byte is received from a slave, assume that the slave is not active and advance to poll the next slave.
4. Otherwise (if a slave is replying), continue to interrupt on EVERY received byte. Subsequent bytes should be received every 520.83mSec. However, since a 1.066mSec Synchronization Delay is required before every console message, a 1.066mSec timeout (reset after every received byte) should be used to detect the end of slave messages.



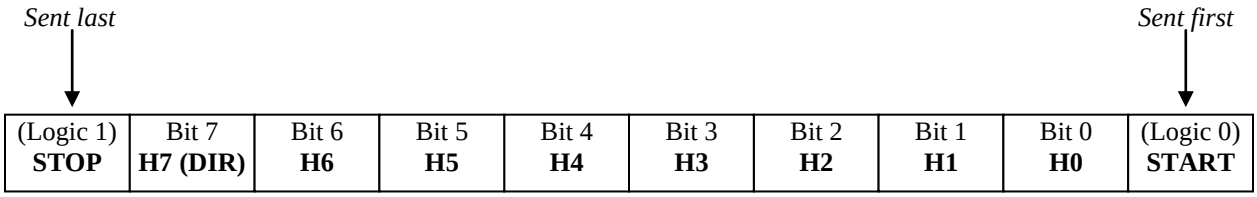
5.0 Message Packet Overview

A message packet is defined as a set of bytes sent by a bus master (console) or slave (speaker). As mentioned, all bits of a message packet must be set as one continuous burst, without interruptions or delays between bytes. Messages of various lengths are supported, but all must follow this Header/Address/Argument/Verifier format:

5.1 Header Byte (1st Message Byte)

All messages on the bus, whether from the console/master or a speaker/slave, must begin with a 1-byte Header (this must always be the first byte transmitted). Bit 7 of this Header is reserved to indicate whether the message is sent by the P2/FedX console or a speaker. **Bit 7 shall be set to 0 for console messages, and set to 1 for speaker messages.** NOTE: the Postman1 Normal protocol messages ALL had Bit 7 set to 0, regardless of the transmitting device, so this represents a change to the format of speaker messages.

Note: as mentioned before, all bytes, including the Header, are transmitted LSB-first. A diagram of the full Header Byte, including the necessary start and stop bits would be

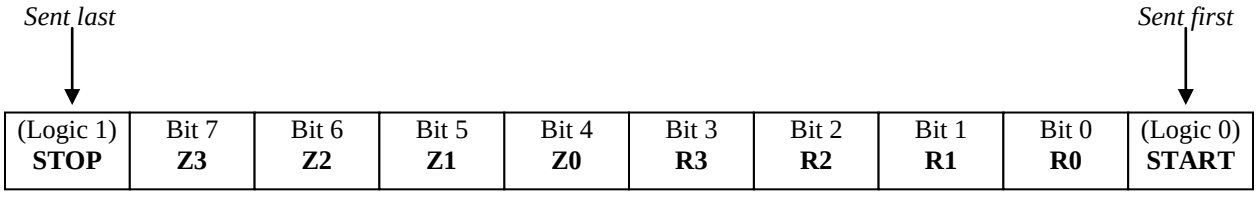


H7..H0 (Bits 7..0): 8-bit Header identifier (see Header Definitions, below). 256 separate commands are therefore possible (128 from the console to speakers, and 128 from speakers back to the console, since bit 7 always must identify the sender). For Postman1, only 17 Headers were actually implemented. Other definitions were suggested, but have been dropped for purposes of this spec since they were never used.

DIR (Bit 7): 1-bit Message Direction Indicator (**0 =sent from the console/master, 1 = sent from speaker/slaves**).

5.2 Console Message Address Byte (2nd Message Byte)

For Smart Speaker messages, the 2nd byte always contains message addressing. for console messages, the format of this byte is identical to that used for Postman1 (NOTE: a slightly different format is defined for speaker messages, below). A diagram of the Address byte would be:



R3..R0 (Bits 3..0): Room Number (0000-1111b). 15 unique rooms allowed (referred to as A-O).
0000b = Room A. 0001-1110b = Room B - RoomO. 1111b = ALL ROOMS, used for broadcasts.

Z3..Z0 (Bits 7..4): Zone Number (0000-1111b). 15 unique zones allowed. Used to identify desired audio stream.
0000b = Zone1. 0001-1110b = Zone2-Zone15. **1111b = ALL ZONES**, **used when console messages should NOT cause speakers to change their input stream/zone selections**.

Examples (neglecting start/stop bits): Zone 1, Room A: 00000000b. Zone 2, Room I: 00011001b.

The zone information contained in the Address byte is passed-through the console from the RF remote control, and identifies for speakers the audio stream that they need to select. For Postman2/FedX, only Zone1 (0000b), Zone2 (0001b) and ALL Zones (1111b) are valid options. All other options are reserved for the future. Options exist to support the day when all rooms can have their own (independent) audio stream.

5.3 Speaker Message Address Byte (2nd Message Byte)

The speaker Address byte defined in the Postman1 Normal protocol was identical to the address byte sent by the console: indicating one of 15 possible zones and one of 15 possible rooms. For Postman2/FedX, the speaker Address byte has been changed to allow speaker replies to indicate whether the speaker OFF, or is listening to (one of its possible) Local sources, or listening to one of the streams sent from the console (Zone1 or Zone2). The format for this byte would be:

Sent last ↓ *Sent first* ↓

(Logic 1) **STOP**	Bit 7 **Z3**	Bit 6 **Z2**	Bit 5 **Z1**	Bit 4 **Z0**	Bit 3 **R3**	Bit 2 **R2**	Bit 1 **R1**	Bit 0 **R0**	(Logic 0) **START**

R3..R0 (Bits 3..0): Same as defined for the console: indicates on of 15 possible rooms, but with 1111b undefined for speakers (since speakers do NOT reply to broadcasts).

Z3..Z0 (Bits 7..4): 0000-1101b: Playing one of 14 possible Zones (0010b=Zone1, 0011b=Zone2, etc.).
1110b: Playing a **local** source.
1111b: **OFF**. Presently playing no local or networked sources.

5.4 Argument Byte(s)

After the Address byte, and before the Verifier byte which ends all messages (see below), are contained the messages Argument byte(s). **Polls do not contain any argument bytes**. Most Smart Speaker messages will contain only 1 Argument byte, but it is allowable to contain any number. See the message definitions, below, for details of the Argument bytes used in various commands. All bytes are sent in the following format:

Sent last ↓ *Sent first* ↓

(Logic 1) **STOP**	Bit 7 **A7**	Bit 6 **A6**	Bit 5 **A5**	Bit 4 **A4**	Bit 3 **A3**	Bit 2 **A2**	Bit 1 **A1**	Bit 0 **A0**	(Logic 0) **START**

A7..A0 (Bits 7..0): 8-bit message arguments. See message definitions.

5.5 Verifier Byte (Last Message Byte)

The final byte of all Smart Speaker messages must be a Verifier byte of the following format. This byte is used to confirm that the basic message information has not been corrupted. As shown in the message definitions below, this byte is generally a software exclusive OR (XOR) of all preceding message bytes, though longer message forms may wish to use this to verify Header/Address only, and include local checksums, etc., within the Argument fields to more robustly verify the bulk of the data payload. See message definitions.

Sent last ↓ *Sent first* ↓

(Logic 1) **STOP**	Bit 7 **V7**	Bit 6 **V6**	Bit 5 **V5**	Bit 4 **V4**	Bit 3 **V3**	Bit 2 **V2**	Bit 1 **V1**	Bit 0 **V0**	(Logic 0) **START**

V7..V0 (Bits 7..0): Generally an XOR of all preceding message bytes. See message definitions.

6.0 Rules for Exchanges

An "exchange" is defined as a set of two bus messages: a message from a console to specific speaker, and the subsequent reply sent from that speaker in return. Here are the rules which govern such exchanges:

6.1 Only the Master Generates Spontaneous Messages

As mentioned, the Postman2/FedX console is the master of all communication on the Smart Speaker bus. For the Postman2/FedX system, what this means is that **only the console is allowed to generate spontaneous transmissions on the bus**.

6.2 The Master Never Interrupts an Exchange in Progress

Regardless of the importance of a message pending with the console/master, outbound transmissions must wait until previously-initiated exchange cycles are completed before initiating a new transmission. Messages in progress either from the console or speakers should never be interrupted.

6.3 A Slave Only Transmits Immediately Following Messages Addressed to It

A slave/speaker is only allowed to transmit immediately following console messages addressed specifically to it (and ONLY it-- **no slave may transmit following a console broadcast**). This rule holds, regardless of the importance of the speaker message pending. It is the console's responsibility to send messages (polls, for example) to each speaker frequently enough to guarantee timely opportunities for speakers to communicate back upstream.

6.4 A Slave Must Reply to Every Message Addressed to It

Unless a speaker is OFF (where replies are optional), a slave/speaker must reply to EVERY console message addressed to it with SOME form of return message (where the default reply must be the Poll_Reply for situations where slaves have nothing significant to communicate at the moment). All replies must be generated within the allowable delay period (767uSec-- see details under Protocol Basics).

In general, speakers may reply to console messages with any one of the following message types:

- a Poll_Reply message (4 bytes). **This will be the default speaker reply, replacing the 1-byte ACK in the Postman1 protocol**, used following all polls, commands, and queries unless higher-priority responses are available to be sent (including query replies).
- a PASS_KEY_CODE Message (see message definitions).
- a DOWNLOAD_INFORMATION Message (see message definitions).

6.5 A Slave Must Reply with the Highest-Priority Information Currently Pending

A speaker/slave may have multiple messages pending when an opportunity comes to transmit back to the console. In this situation, the highest-priority message should be sent. The general guidelines for possible message types are as follows:

6.5.1 Highest Priority: Non-Poll Query Replies

If a console has sent a specific query message (other than a "poll"-- see message definitions) to a speaker, the speaker must send the proper reply as soon as possible. Lower-priority replies can be sent until the query reply is assembled, but, once assembled, it must be given the highest priority in the outbound queue.

6.5.2 Medium Priority: Pass_Key_Code Messages

Some buttons pressed on the speaker/slave products (or their local remote controls) may require urgent changes to the console audio sources, etc. A Pass_Key_Code message is defined below to transport appropriate presses to the console. These messages should be given a medium priority in the slaves' outbound queues (they should pend when query replies are available in the queue, but take precedence over poll replies).

6.5.3 Medium Priority: Download_Information Messages

A Download_Information message is defined below to allow speaker/slaves to communicate blocks of data up to the console. It is expected that this data would typically represent information related to local sources being played on the slave (Ballpark's current AM/FM station, for example). Such messages are given a medium priority, identical to Pass_Key_Code messages. They should therefore be buffered in a FIFO with Pass_Key_Code messages, taking precedence over poll replies, but pending when all other specific query replies are in the queue.

6.5.4 Lowest Priority: Poll Replies

The default reply, referred to as the Poll_Reply (see message definitions) should be sent when no other speaker message is in the outbound Smart Speaker queue. This reply contains only basic information concerning the speaker's on/off state and its volume level, deemed less important than the other classes of replies.

6.6 A Master Continuously Polls All Speakers, but as a Low Priority

To allow speakers to send information upstream to the console in a timely manner, polling messages like those used in Postman1 (previously based around ON/OFF_STATUS Queries, but re-defined around a new POLL message) must be sent more frequently, and will be used as an opportunity to keep aware of speaker status changes. As with all console messages, speakers may optionally reply to polls with any of the message types listed above. Full timing details of the new console polling cycle are described below. Generally, these polls shall be sent continuously, but only while no higher-priority messages are available in the console's outbound queue.

6.7 A Master Must Transmit the Highest-Priority Message Currently Pending

As with slaves, the console/master outbound messages are assigned priorities. Higher-priority messages must be given precedence over those with lower priorities, as follows:

6.7.1 Highest Priority: Non-Poll Queries and Control Messages

If a query or control message needs to be sent to specific speaker, it takes highest priority. See message definitions.

6.7.2 Medium Priority: Pass_Key_Code Messages

The Postman2/FedX console is capable of passing-through RF commands unused by the console directly to the speaker. The Pass_Key_Code message, defined below, will be used for this. Such messages will have a lower priority than queries, but a higher priority than polls. When a Pass_Key_Code message is pending, the polling cycle will be immediately interrupted (after any exchange in progress has completed), the Pass_Key_Code message will be sent, and then polling can resume.

6.7.3 Medium Priority: Download_Information Messages

The Postman2/FedX console is capable of downloading larger blocks of data to speakers using the Download_Information message (see message definitions). This may be used for updating speaker application code, etc. The priority for these messages will be lower than queries, but higher than polls. As with Pass_Key_Code messages, the polling cycle must be interrupted temporarily to send any pending Download_Information message.

6.7.4 Lowest Priority: Polls

Speaker polls are assigned the lowest priority, and should represent a continuous cycle of messages sent while the console has no more important information to transmit. Sending these polls guarantees each speaker a regular opportunity to send messages upstream to the console.

6.8 Only One Speaker Will Be Polled During a Query Exchange

When Postman1 sent a query, Cobalt2 would generate its reply as soon as possible, but reply delays were not specifically controlled. For P2/FedX, a full Query exchange between the console and a speaker will be redefined to take place as follows:

1. When a query needs to be sent, the console **interrupts the normal polling cycle** (after any exchange in progress completes) and immediately sends the desired query to the appropriate speaker.

2. The speaker sends its reply within 767uSec, if possible. If not, it replies with a POLL_RESPONSE message within 767uSec.
3. The console withholds polling all other speakers, and instead **polls this speaker only**, every 6mSec. The speaker replies with POLL_RESPONSE messages until its query reply is ready. Once ready, it replies with the desired query response.
4. After receiving the desired query response, the console resumes the normal polling cycle for all speakers. Alternately, if a speaker fails to reply at all, or replies as OFF, normal polling can also resume, at the discretion of higher-level software.

6.8 Full List of Commands Used Outside the Main Room

Cobalt2 (the speaker in the "Main Room") will likely continue to use the full set of Smart Speaker messages defined for Postman1 (see message definitions, below). Speakers outside the Main Room will likely need only limited control: volume up/down, mute/unmute, and on/off, for example. All other specific controls for these speakers may be handled by passing remote control commands transparently through the console to the speaker (using the Pass_Key_Code command), where they will be interpreted properly.

The following shall be the full list of commands able to be received by non main-room speakers. See message definitions, later, for details of constructing the messages:

6.8.1 POLL and POLL REPLY Messages (Headers 0x00/0x80)

Polls are the lowest-priority message, sent continuously by the console to all speakers as determined by the polling cycle. Used by the console to continuously monitor the ON/OFF, Local/Network and volume/mute status of all speakers, as well as to provide frequent opportunities for speakers to send messages to the console. Interrupted whenever a higher-priority message needs to be sent. Speakers typically reply with a Poll Reply, but must substitute higher-priority replies if any are pending.

6.8.2 ON/OFF Message (Header 0x01)

Ultimately, this may only be used by the console to turn speakers OFF. Speakers may turn on after receiving a number of different passed-through RF remote buttons, or after receiving a command from a local IR remote. In these cases, polling will be used to detect its power-on.

6.8.3 PASS_KEY_CODE Message (Headers 0x0D/0x8D)

As described, this may be used by the console (using Header 0x0D) to pass speaker-specific RF remote control commands through. The console may not interpret these commands at all. Using Header 0x8D, speakers will be capable of sending button presses back upstream to the console.

6.8.4 SET_MAIN_ATTENUATION Message (Header 0x02)

Here, the console may ultimately only use the Mute and Return_to_Last_Volume arguments, as broadcasts, in response to Mute All events.

6.8.5 QUERY_SPEAKER_INFO Message (Header 0x0B)

Used by the console to determine the variety of speaker attached. Depending on the architecture of Hermes, the console also may also use the Query_Main_Attenuation and Query_Download_Info_Status arguments. This may expand if AdaptIQ gets defined for non main-room speakers.

6.8.6 DOWNLOAD_INFORMATION Message (Headers 0x0A/0x8A)

This message (Header 0x0A) may be used to push Postman2/FedX console status information onto the Ballpark VFD, or may be used for sending other data blocks to a speaker (possibly application code updates, etc.). Using Header 0x8A, speakers can use this message type to upload blocks of data to the console (source-related data needed to be displayed on Hermes, for example).

7.0 The Polling Cycle

7.1 Overview

A polling structure like that already supported by the Normal protocol's QUERY_SPEAKER_INFO/ ON_OFF_Status message will be used to keep in continuous contact with networked speakers. A new POLL message will be used for his purpose, however. The standard replies to these polls will provide basic speaker status information regarding its on/off state, local/network listening mode, its mute state and its volume level.

However, speakers will have the ability to substitute higher-priority messages (if pending) for its standard poll reply, providing opportunities to transfer important button presses and blocks of data back upstream to the console.

Speakers connected to the network may be off for long periods of time. When off, speakers only need to be polled occasionally to detect when a keystroke has turned them on. However, speakers which are both connected and turned on need to be polled rapidly to guarantee good response time for messages sent upstream to the console. A Polling Cycle is therefore proposed which meets these needs, and whose performance is controlled to degrade predictably but gracefully as the number of ON speakers increases.

7.2 Polling Cycle Timing Details

The polling process organizes all 15 possible speakers into two lists: presently ON speakers, and NOT ON speakers (off or disconnected). The console first polls all members of the presently ON speakers list (up to 15, in sequence), then polls a single member of the NOT ON list (if any). This represents one subcycle of the Polling Cycle. Then the subcycle repeats (with no interruption), using the same list of ON speakers but the next member of the NOT ON list (in a circular, round-robin fashion).

For example, if only speaker A is ON, the total Polling Cycle would be:
AB, AC, AD, AE, AF, AG, AH, AI, AJ, AK, AL, AM, AN, AO, then this would repeat, without interruption. 14 total subclass.
One subcycle ~ 11mSec. Total cycle ~ 153mSec.

If only speakers C and G are known to be ON, the Polling Cycle would be:
CGA, CGB, CGD, CGE, CGF, CGH, CGI, CGJ, CGK, CGL, CGM, CGN, CGO, then this would repeat. 13 subcycles.
One subcycle ~ 16mSec. Total cycle ~ 214mSec.

If all speakers were ON, the Polling Cycle would be:
ABCDEFGHIJKLMNO, then this would repeat. 15 subcycles.
One subcycle = total cycle ~ 82mSec. This results in the longest latency for messages from speakers to the console.

If all speakers were NOT ON, the Polling Cycle would be the same as for all ON. 15 subcycles.

Longest total cycle would be any 7 speakers ON, the rest NOT ON.
One subcycle ~ 44mSec. Total cycle ~ 351mSec. This has the longest latency identifying speakers turning ON.

When each SUBCYCLE of the Polling Cycle is completed, the list of ON and NOT ON speakers should be updated based on poll responses. Any previously NOT ON speaker which replied as ON in the last subcycle should be moved immediately to the ON list. All previously ON speakers which replied as OFF or have failed to reply all together for 5 subcycles should be moved to the NOT ON list. When polling first begins after console reset, all speakers begin in the NOT ON list.

Number of ON Speakers	Poll Period for a Given ON Speaker (One Subcycle)	Poll Period for ALL Speakers (Total Polling Cycle)
0	n.a.	82 mSec
1	11 mSec	153 mSec
2	16 mSec	214 mSec
3	22 mSec	263 mSec
4	27 mSec	301 mSec
5	33 mSec	329 mSec
6	38 mSec	345 mSec
7	44 mSec	**351 mSec**
8	49 mSec	345 mSec
9	55 mSec	329 mSec
10	60 mSec	301 mSec
11	66 mSec	263 mSec
12	71 mSec	214 mSec
13	77 mSec	153 mSec
14	82 mSec	82 mSec
15	**82 mSec**	82mSec

7.3 Typical Poll and Reply Message Bytes

Unless the console or speaker has higher-priority messages to send, the system will default to a polling cycle using messages formatted as follows:

Console Poll (**3 bytes**):

First (Header) byte: 0x00 (POLL message).
Second (Address) byte: as per the present Normal protocol.
Third (Verifier) byte: as per the present Normal protocol.

Speaker Poll Reply (**4 bytes**):

First (Header) Byte: 0x80 (POLL REPLY).
Second (Address) byte: Speaker address byte, including on/off/local/network status.
Third (Argument) byte: Becomes volume/mute status, as follows:
Argument Details:
Bits 6..0: Speaker's Main Attenuation level, per the present Normal protocol.
Bit 7: Mute state: 1=muted. 0=not muted.
Fourth (Verifier) byte: as per the present Normal protocol.

7.4 Using the PASS_KEY_CODE Message as a Reply (Header 0x8D

As is true following ANY console message, this (4-byte) message must, if available to be sent, be returned **instead of the SPEAKER_INFO reply** (0x8C), following a console POLL message. The Argument byte will contain the key code identifier associated with one of 256 possible buttons meant to control the console. NOTE: existing key codes for IR/RF remotes should be used, if possible.

7.5 Using the DOWNLOAD_INFORMATION Message (0x8A) as a Reply

This (N-byte) reply must, if available to be sent, be returned instead of the SPEAKER_INFO reply (0x8C) following any console message, including polls. The Argument/Data bytes could contain any sort of data, high-level commands or queries (all presently undefined) intended for the console. This may be used for sending local source-related data to the console for display on the Hermes remote. It would conceptually also allow a speaker to directly command source changes, etc., or to request zone/source-specific status information for a local display (tuner station being played, etc.), as Couchboy did.

8.0 High-Level Speaker Control Issues

8.1 Master/Slave Configuration

As with Postman1, all communications on the P2/FedX Smart Speaker bus are controlled by the bus Master. **There may be only one Master on a given bus.** The bus Master may initiate a transmission at any time, as long as the bus is Idle (it must allow all messages and replies currently in progress to complete before initiating transmissions), and as long as it respects the message timing details described later. **For the P2/FedX system, the P2/FedX console is always the bus Master.**

Speakers act as slaves on the Smart Speaker bus**, and never generate spontaneous transmissions. They may only transmit immediately following a message sent to them by the console.** However, as described later, ANY console message may be used as an opportunity for speakers to initiate transmissions.

8.2 Room Addressing Must be Unique

New speakers added to the network must have a unique 4-bit Room Address (RoomA through RoomO, matching the remote control intended to operate the speaker) properly set before connecting to the network.

NOTE: if two speakers are connected to the network with the same Room Address, their replies will collide on the bus, interfering with the proper operation of both speakers (and potentially slowing the response of other speakers). Once all connected speakers have unique Room Addresses, any control problems should end.

8.3 Managing Separate Physical Buses for Zone1 and Zone2

The P2/FedX console has two separate physical Smart Speaker buses (one for each Speaker Output connector). It is expected that Cobalt2 will be the only Smart Speaker ever plugged into the Zone1 Speaker Output connector. Cobalt2 is shipped as RoomA (and typically would never be changed in the field, although nothing prevents it).

Both Zone1 and Zone2 audio streams are available on the Zone2 Speaker Output connector, as is the Zone2 Speaker Data signal. Therefore, it is most likely that RoomB through RoomO speakers will be plugged into the Zone2 Speaker Output connector, and controlled using the Zone2 Speaker Data signal. **LSA1 (the only existing speaker to be supported by P2/FedX which speaks the "Legacy" protocol) must be connected to the Zone2 Speaker Output. Note: LSA1 cannot co-exist with new speakers on the same network.**

This means that the two possible network topologies for P2/FedX will be:
1. Cobalt2 (RoomA) plugged into the Zone1 Speaker Output, communicating via the protocol in this document. Rooms B through O plugged into the Zone2 Speaker Output connector, controlled in the same way. Zone2 protocol set to "Normal" via the OSD. An AM5P/AM20P or 901P may be connected to the Zone2 variable output (controlled as RoomO).
2. Cobalt2 (RoomA) plugged into the Zone1 Speaker Output, communicating via the protocol in this document. An LSA1 plugged into the Zone2 Speaker Output connector. Zone2 protocol set via the OSD to be "Legacy". No new Smart Speakers may be connected to Zone2, due to the incompatibility of the Legacy protocol. An AM5P/AM20P or 901P may be connected to the Zone2 variable output (controlled as RoomO).

8.4 Speaker Control Modes

Smart Speaker messages are defined which allow **two different modes of controlling the speaker**: Master Mode and Pass-Through Mode. Master Mode commands allow a controlling unit (e.g., the Postman console) to keep track of all critical speaker parameters, and to send these as absolute values (arguments) to the speaker at the appropriate time. Pass-Through Mode allows a less intelligent controlling unit (as simple as an IR/Smart Speaker protocol bridge) to effectively pass remote control messages directly to the speaker. In this mode, the speaker keeps track of critical parameters (its last volume level before being muted, etc.).

Consoles wishing to control networked speakers via Master Mode are required to maintain a table describing the available features of each type of speaker to be controlled (types and ranges of controls, etc.).

8.5 Detecting New Speakers

After powering-up (hardware reset), networked speakers are responsible for entering the OFF STATE (muted, with hardware in its lowest-current state. In this state, only the hardware responsible for interfacing with the network is required to be operational. The speaker must be able to receive and reply to (at a minimum) Poll Messages, Query_Speaker_Info Messages, as well as ON/OFF Messages.

Since the console continuously polls all possible speaker Room Addresses, a speaker newly-added to the Smart Speaker bus can be detected as soon as it begins replying to polls. Once a new speaker has been detected, the console should send a Query_Speaker_Info message to identify its type.

8.6 Powering-Up Networked Speakers

Before sending a networked speaker any commands other than Poll or Query_Speaker_Info messages, it should first be turned on using an ON/OFF Message. This allows the speaker's hardware to become fully operational. The console should delay appropriately to allow this power-up process to complete. If a table is not available describing the required power-up delay for each networked speaker, consoles should use a default delay time of 1 second.

8.7 Detecting Speaker Status Changes Through Polling

During the course of polling, bits 7..4 of the Address Byte of the speaker's replies will contain information regarding its on/off status, as well as the source it's playing (Zone1, Zone2 or a local source of its own), as follows:

Z3..Z0	(Bits 7..4):	0000-1101b:	Playing one of 14 possible Zones (0010b=Zone1, 0011b=Zone2, etc.).
		1110b:	Playing a **local** source.
		1111b:	**OFF**. Presently playing no local or networked sources.

By monitoring these bits, the console can detect when the following important status changes occur:

8.7.1 Speaker Turn On/Off

Speakers such as Ballpark and Knex could be turned on and off by means of buttons on the units themselves, or via their IR remote controls, or via Hermes remote control commands which are simply passed-through the console without an understanding of their significance. Using the Address Z3..Z0 bits, the P2/FedX console can monitor the on/off state of each speaker. This allows the console to power-off when all speakers are off, for example.

8.7.2 Speaker Source Changes

Similarly, speakers can switch between Zone1 and Zone2 (or a local source) without the console anticipating it. By monitoring the same Address bits, the console can detect such source changes and take appropriate action (such as powering up/down the appropriate audio paths).

8.8 Managing Speaker Stream Switching

Speakers such as Ballpark, Knex and SA2/3 will have the ability to select either the Zone1 or Zone2 networked analog outputs from the console. The speaker will make this selection automatically, by reacting to the Zone bits in the Address byte of every console Pass_Key_Code message (Header 0x0D) and On/Off message (Header 0x01). These messages are sent to speakers as a direct result of button presses on its RF remote control, and will therefore have their Zone bits set exactly as received in the message from the remote (these bits are passed directly though from the remote to the speaker). Note: On/Off messages are included here because the console must send one in response to all remote control source button presses (AM/FM, AUX, CD/DVD, etc.), where the On/Off message ensures that networked speakers are both ON and unmuted when a source is selected. **Note also: Pass_Key_Code messages sent to speakers playing Local sources will NOT switch speakers away from Local mode.**

A Pass_Key_Code message with an On/OFF button argument passed to speakers allows them to power-up in their last-played source (either Network or Local). If a speaker chooses to play a networked stream/zone in this, case, it must select the stream indicated in the message Address (as opposed to the last-played stream).

9.0 Message Definitions

9.1 Quick Reference Tables

Messages Sent By: CONSOLE/MASTER		Received By: SPEAKERS/SLAVES		
Section	**Message**	**Header Byte**	**Total Bytes**	**Queue Priority**
	POLL MESSAGE	0x00	3	Low
	ON/OFF MESSAGE	0x01	4	High
	SET MAIN ATTENUATION	0x02	4	High
	SET SECONDARY LEVELS	0x03	4	High
	SET EQ TYPE / TONE LEVELS	0x04	4	High
	SET SPEAKER MODE	0x05	4	High
	CONTROL EFFECTS	0x06	4	High
	SELECT INPUT SOURCE	0x07	4	High
	SELECT DECOMPRESSOR	0x08	4	High
	SELECT POST PROCESSOR	0x09	4	High
	DOWNLOAD INFORMATION	0x0A	5-255	Medium
	QUERY SPEAKER INFO	0x0B	4	High
	unused	0x0C		
	PASS KEY CODE	0x0D	4	Medium
	unused	0x0E-10		
	INSTALLER SERVER PUSH	0x11	4	High
	INSTALLER SERVER EXEC	0x12	4	High
	unused	0x12-0x7F		

Messages Sent By: SPEAKERS/SLAVES		Received By: CONSOLE/MASTER		
Section	**Message**	**Header Byte**	**Total Bytes**	**Queue Priority**
	POLL REPLY	0x80	4	Low
	unused	0x81-89		
	DOWNLOAD INFORMATION	0x8A	5-255	Medium
	unused	0x8B		
	QUERY SPEAKER INFO REPLY	0x8C	4-9	High
	PASS KEY CODE	0x8D	4	Medium
	unused	0x8E-92		
	QUERY INSTALLER SERVER REPLY	0x93	4	High
	unused	0x94-0xFF		

9.2 Messages Sent by the Console

9.2.1 POLL Message (Header 0x00)

Sender: **Console**
Total Bytes: **3**
Queue Priority: **LOW**

The Poll Message is the **only 3-byte message presently defined**. The 4 most-significant bits of its Address byte identify the proper audio Zone for the speaker to play. The 4 least-significant bits identify the speaker being polled (by its Room Number, as set by the speaker DIP switches).

No Argument is required, so it is left out, to preserve network bandwidth.

Poll messages are sent, in effect, as a simple form of query to obtain Poll Response information, and to provide speakers a regular opportunity to pass any other important messages.

Byte	Name	Description
1	Header	**0x00**
2	Address	General description: b7..b4: Zone Number (0000-1111b). 15 unique zones allowed. 0000b = Zone1 (passed-through from the RF remote control). 0001b = Zone2 (passed-through from the RF remote control). 0010b-1110b = unused by Postman2/FedX. Reserved for future zones. 1111b = All zones. Used for broadcasts, such as Mute All assert. b3..b0: Room Number (0000-1111b). 15 unique rooms allowed (referred to as A-O). 0000b = Room A 0001-1110b = Room B - RoomO. 1111b = ALL ROOMS, and is only used for messages broadcasted to all rooms of the designated zone.
-	Argument	No Argument sent for this message. 3 total message bytes only.
3	Verifier	XOR of bytes 1 and 2.

POLL Message

9.2.2 ON/OFF Message (Header 0x01)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

An ON/OFF Message informs the addressed Smart Speaker(s) to fully power-up in preparation for playing audio, or to power down after a session. Note: this command will be used by the console when source buttons are pressed on the remote, to ensure that a speaker is fully powered up, unmuted, and selecting the desired network input stream.

Byte	Name	Description
1	Header	**0x01**
2	Address	Same as for the console POLL Message.
3	Argument	0x00 = Begin normal power-up procedure, but remain **MUTED**. 0x01 = Power-up normally and ramp to the last state before OFF (last Master Attenuation level, Secondary Attenuation levels, EQ Type, Speaker Mode, Tone Levels, Effects states, etc**.). Select the NETWORK input stream (zone) identified in the Address Byte**. NOTE: if already ON but MUTED, this command will UNMUTE. 0x02-0x7F = undefined. 0x80 = Power-down normally/slowly (ramp volume, soft mute, etc.). 0x81 = Power-down immediately (power fail imminent). Immediate mute. 0x82-0xA9 = undefined. 0xAA = Power up in TAP protocol mode (used by manufacturing). 0xAB-0xEF = undefined. 0xF0 = Execute software RESET (useful after downloading new code, etc.). 0xF1-0xFE = undefined. 0xFF = Toggle between ON (power-up and ramp to last state) and OFF (power-down normally/slowly).
4	Verifier	XOR of bytes 1 through 3.

ON/OFF Message

It is expected that some amount of time may be necessary for the speaker's power supply rails, etc., to settle before being able to unmute (or, possibly, before the speaker is ready to accept another command). After being turned on, the system can address this issue in three ways:

1. The speaker can be queried for readiness by the console (using the *QUERY SPEAKER STATUS* command, *ON/OFF Status* argument) before being commanded to configure its speaker mode, attenuation levels, etc.

2. The speaker can simply choose to not reply to subsequent *SET SPEAKER MODE*, etc., commands until it is ready to unmute.

3. The console can institute a delay (typically about 1 second) between sending the On/OFF message and sending any subsequent messages.

9.2.3 SET MAIN ATTENUATION Message (Header 0x02)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

A Set Main Attenuation message commands the addressed Smart Speaker(s) to change their master volume setting to the level indicated. NOTE: the argument contains the new level, expressed as dB of attenuation, where 0dB must always represent a speaker's maximum volume. A speaker's available dynamic range is passed through the Query Speaker Info (Speaker Type) reply.

Byte	Name	Description
1	Header	**0x02**
2	Address	Same as for the console POLL Message. **Mute All Assert and De-assert set all zone/room bits.**
3	Argument	b7=1: Ramp to the desired master attenuation level, using the speaker's internal "slow" volume ramp capability, as follows: • If the desired volume is >30dB louder than the speaker's present level, the speaker shall immediately go to a level 30dB below (more attenuated than) the desired attenuation level. It shall then, under its own control, ramp its master attenuation to the desired level at a rate of 1dB per 70mSec, +/- 10%. • If the desired speaker volume is <30dB louder than its present level, it shall immediately begin ramping from its present level, at the 70mSec/dB rate. • If the desired volume is quieter than the speaker's present level, the speaker shall immediately begin ramping toward the desired level at the 70mSec/dB rate. If the desired level is more than 30dB from the starting level, the speaker shall only ramp for the first 30dB, and then jump immediately to the desired level. b7=0: Go immediately to the new master attenuation level described (no ramp). Speakers, optionally, should include a "fast" ramp here (reaching the desired level in 50mSec). b6..b0 = Set the desired master attenuation level for the speaker (expressed as dB of attenuation): 0000000b = no attenuation (full volume). Speaker's loudest level. 0000001b = 1dB attenuation. 0000010b = 2dB attenuation, etc. 1110111b = 119dB attenuation. NOTE: attenuation levels greater than those supported by the speaker shall result in the speaker going to MUTE. ***SPECIAL EXCEPTIONS:*** 1111000b = **MUTE** (go to maximum possible attenuation). 1111001b = **UNMUTE** (return to last volume level before OFF or MUTE. This level is stored by the speaker). 1111010b = Toggle between MUTE (1111000b) and UNMUTE (1111001b). 1111011b = **VOLUME UP 1dB** (reduce attenuation 1dB) from present level. 1111100b = **VOLUME DOWN 1dB** (increase attenuation 1dB) from present level. 1111101b = **MUTE ALL ASSERT** (Mute. Set Mute All Active flag if previously unmuted). 1111110b = **MUTE ALL DE-ASSERT** (Unmute if Mute All Active flag set). 1111111b = undefined. Examples: Mute immediately (no ramp): 01111000b. Unmute with ramp to volume level "Max-12" (12dB attenuation): 10001100b. Ramp to volume level “Max-6” (6dB of attenuation): 10000110b. Increase volume by 1dB (decrease attenuation by 1dB): 01111011b.
4	Verifier	XOR of bytes 1 through 3.

SET MAIN ATTENUATION Message

9.2.4 SET SECONDARY LEVELS Message (Header 0x03)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x03**
2	Address	Same as for the console POLL Message.
3	Argument	b7..b5 select one of 8 secondary levels to adjust: 000 = Center Level. 001 = undefined. 010 = Surround Level. 011- 111 = undefined. b4..b0 select the desired absolute level: 00000-01110b = steps below/left/front of default (down to -15 steps). 10000b = **"speaker default."** 10001-11110b = steps above/right/rear of default (up to +14 steps). ***NOTE**: a "step" size may or may not correspond to a change of 1dB-- to be determined by the speaker being controlled.* ***SPECIAL EXCEPTIONS:*** 01111b **= INCREMENT 1 STEP FROM PRESENT LEVEL.** 11111b **= DECREMENT 1 STEP FROM PRESENT LEVEL.** NOTES: Couchboy/Cobalt I: Center has +/- 8 steps, surround has +6/-10 steps. Cobalt II: Center has +/- 8 steps, implemented as follows: +8: speaker increases center gain by 6dB. +7: speaker increases center gain by 5dB. +6: speaker increases center gain by 4dB. +5: speaker increases center gain by 3dB. +4: speaker increases center gain by 2dB. +3: speaker increases center gain by 1dB. +2: speaker oozes 0% into L/R, flat center gain. +1: speaker oozes 10% into L/R. 0: speaker oozes 20% into L/R. (default.) -1: speaker oozes 30% into L/R. -2: speaker oozes 40% into L/R. -3: speaker oozes 50% into L/R. -4: speaker oozes 60% into L/R. -5: speaker oozes 70% into L/R. -6: speaker oozes 80% into L/R. -7: speaker oozes 90% into L/R. -8: speaker oozes 100% into L/R. Surround has +/- 10 steps (implemented as 1dB/step). Examples: Set Center level 3 steps above default: 00010011b. Set Surround level 6 steps below default: 01001010b. Decrement Surround level 1 step from present level: 01011111b.
4	Verifier	XOR of bytes 1 through 3.

SET SECONDARY LEVELS Message

9.2.5 SET EQ TYPE/TONE LEVELS Message (Header 0x04)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x04**
2	Address	Same as for the console POLL Message.
3	Argument	b7..b5 selects an overall EQ type, OR select one of 7 Tone Levels to adjust, as follows: 000b = Set Overall EQ Type (types defined in b4..b0, see below). 001b = Set Treble Level. 010b = Set Bass Level. 011b - 111b = undefined. When setting overall EQ type (b7..b5 = 000b), b4..b0 define the type as follows: 00000b **= NEXT EQ TYPE (scroll to next type in sequence).** 00001b = Audio (Normal) EQ Mode. 00010b = Film EQ Mode. 00011b = Audio (Normal) EQ Mode, but add 1-sample delay to right channel. 00100b = Film EQ Mode , but add 1-sample delay to right channel. 00100-11111b = undefined. Otherwise, when adjusting a tone level, b4..b0 select the desired level, as follows: 00000-01110b = steps (usually dB) below speaker's default (down to -15 steps). 10000b = "speaker default." 10001-11110b = steps (usually dB) above speaker's default (up to +14 steps). ***SPECIAL EXCEPTIONS:*** 01111b **= INCREMENT 1 STEP (USUALLY 1dB) FROM PRESENT LEVEL.** 11111b **= DECREMENT 1 STEP (USUALLY 1dB) FROM PRESENT LEVEL.** NOTES: Couchboy/Cobalt I: had no software tone control capability. Cobalt II: bass/treble range = +/- 15 steps before Installer. After Installer, bass range is +3/-6 steps and treble range is +/- 0 steps (not adjustable). Examples: Select Next EQ Type (in Sequence): 00000000b. Select Audio (Normal) EQ Type: 00000001b. Select Film EQ Type: 00000010b. Set Bass level 3dB above default: 01010011b. Set Treble level 6dB below default: 00101010b. Decrement treble 1dB from present level: 00111111b.
4	Verifier	XOR of bytes 1 through 3.

SET EQ TYPE / TONE LEVELS Message

9.2.6 SET SPEAKER MODE Message (Header 0x05)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x05**
2	Address	Same as for the console POLL Message.
3	Argument	Speaker mode desired, as follows: 0x00 **= NEXT SPEAKER MODE (scroll to speaker's next possible speaker mode, in sequence)**. 0x01 = **"BEST" SPEAKER MODE (speaker-defined mode best suited to this source).** 0x02 = stereo (left and right only). 0x03 = stereo + center. 0x04 = undefined. 0x05 = stereo + center + left and right surrounds (for video sources). 0x06-0xFF = undefined.
4	Verifier	XOR of bytes 1 through 3.

SET SPEAKER MODE Message

9.2.7 CONTROL EFFECTS Message (Header 0x06)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x06**
2	Address	Same as for the console POLL Message.
3	Argument	b7..b4 select 1 of 16 effects to control, as follows: 0000b = Control Dynamic Range Compression (DRC). 0001b = Control Mono-to-6 Boingerizer. 0010 = Control Installer. 0011-1111b = undefined. When controlling an effect, b3..b0 select the desired state, as follows: 0000b = Disable the effect. 0001b = Enable the effect. 0010b = Toggle the effect. 0011-1111b = undefined. ***SPECIAL CONTROLS for INSTALLER***: 0011b = Write System Parameters to FLASH (may be redundant). 0100b = Transfer Coefficients from EQs to system parameter block. 0101b = Uninstall the system. Both 0100b and 0101b should be followed by a 0011b call to make the change permanent. 0x05 is required because after the Installer process, the coefficients end up in the EQs and the sequence 0100b, 0011b makes them permanent. Examples: Enable Dynamic Range Compression: 00000001b. Disable Mono-to-6 Boingerizer: 00010000b. Toggle Installer On/Off: 00100010b.
4	Verifier	XOR of bytes 1 through 3.

SELECT EFFECTS Message

9.2.8 SELECT AUDIO INPUT Message (Header 0x07)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x07**
2	Address	Same as for the console POLL Message.
3	Argument	The Argument field selects one of the speaker's inputs to play (NOTE: All input types defined here may not apply every speaker), as follows: 0x00 **= NEXT INPUT (select next available speaker input, in sequence).** 0x01 = Select speaker's primary (Console) analog L/R input. 0x02-0x0F = Select speaker's auxiliary analog inputs. 0x10 = Select speaker's primary (Console) S/PDIF input. 0x11-0x1F = Select speaker's auxiliary S/PDIF inputs. 0x20 = Select speaker's first local source. 0x21-0x2F = Select speaker's other local inputs. 0x30-0xFF = undefined.
4	Verifier	XOR of bytes 1 through 3.

SELECT AUDIO INPUT Message

9.2.9 SELECT DECOMPRESSOR Message (Header 0x08)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x08**
2	Address	Same as for the console POLL Message.
3	Argument	0x00 = none (PCM) 0x01 = AC-3 0x02 = MPEG-2 0x03 = AAC 0x04 = DTS 0x05 = MP-3 0x06 = unknown (to be identified by the speaker, if possible). 0x07 = AC-3, Dolby 1+1 Mode (ACMOD = 0), play Left Channel Only. 0x08 = AC-3, Dolby 1+1 Mode (ACMOD = 0), play Right Channel Only. 0x09 = AC-3, Dolby 1+1 Mode (ACMOD = 0), play both Left and Right. 0x0A = AC-3, Dolby 1+1 Mode (ACMOD = 0), play Left + Right. 0x0B-0xFF = undefined.
4	Verifier	XOR of bytes 1 through 3.

SELECT DECOMPRESSOR Message

9.2.10 SELECT POST PROCESSING Message (Header 0x09)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x09**
2	Address	Same as for the console POLL Message.
3	Argument	0x00 = **SELECT NEXT POST PROCESSING ALGORITHM (in sequence).** 0x01 = Select NO post processing (play audio as 2-channels). 0x02 = Select Videostage to matrix-decode multiple channels. 0x03 = Select Dolby Digital to matrix decode multiple channels. 0x04 = Select "Audiostage," to matrix decode multiple channels. 0x05 – 0xFF = undefined.
4	Verifier	XOR of bytes 1 through 3.

SELECT POST PROCESSING Message

9.2.11 DOWNLOAD INFORMATION Message (Header 0x0A)

Sender: **Console**
Total Bytes: **5-255**
Queue Priority: **MEDIUM**

The DOWNLOAD_INFORMATION message was defined for Postman1 and used for various purposes. Its format allows up to 250 bytes of data to be sent to a speaker via the Smart Speaker packet itself, where the payload can be defined as desired (including local checksum, etc.). Variants of this message type could be defined to write directly to speaker VFD's, or to announce changes to console status (new source being played, new track/station, etc.).

Byte	Name	Description
1	Header	**0x0A**
2	Address	Same as for the console POLL Message.
3	Argument	0x00 = Download Console Type (Data fields TBD) 0x01 = Download Console Software Version (Data fields TBD) 0x02 = Download Console Serial Number (Data fields TBD) 0x03-0x0F = undefined. 0x10 = Download Source Change Data Block (Input Source, Speaker Mode, Effects, Decompressor, and Post Processing bytes follow as Data fields. Length is 10 bytes.) 0x11-0x1F = undefined. 0x20 = Download New Speaker EQ (data contained in this message). 0x21 = Download New Speaker EQ (data contained in S/PDIF stream). 0x22-0x2F = undefined. 0x30 = Download New Application Code (data contained in this message). 0x31 = Download New Application Code (data contained in S/PDIF stream). 0x32-0xFE = undefined. 0xFF = Abort Download.
4	Length	Length of the complete message, in bytes, including the Header, Argument and Checksum fields. 0x00 - 0x04 = illegal. 0x05 - 0xFF = length of entire message, in bytes.
5-254	Data	8-bit data bytes.
Last	Verifier	XOR all previous message fields, including Header.

DOWNLOAD INFORMATION Message

9.2.12 QUERY SPEAKER INFO Message (Header 0x0B)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x0B**
2	Address	Same as for the console POLL Message.
3	Argument	0x00 = Query speaker ON/OFF status (Used for polling networked speakers). 0x01 = Query speaker MAIN ATTENUATION status. 0x02 = Query speaker SECONDARY LEVELS status. 0x03 = Query speaker TONE LEVELS status. 0x04 = Query speaker SPEAKER MODE status. 0x05 = Query speaker AUDIO INPUT status. 0x06 = Query speaker DECOMPRESSOR status. 0x07 = Query speaker POST PROCESSING status. 0x08 = Query speaker DOWNLOAD INFO status. 0x09 = Query speaker RUNTIME INSTALLER status 0x0A - 0x0F = undefined. 0x10 = Query speaker Type. 0x11 = Query speaker Software Variant (used for variations within product families). 0x12 = Query speaker Software Revision. 0x13 = Query speaker Serial Number. 0x14-0xEF = undefined. 0xFX = Query speaker EFFECT status, where X is a 4-bit value identifying the effect (1 out of 16 possible) whose status is requested. 0xF0 = DRC status. 0xF1 = Boingerizer status. 0xF2 = Installer status. 0xF3-0xFF = undefined.
4	Verifier	XOR of bytes 1 through 3.

QUERY SPEAKER INFO Message

9.2.13 PASS KEY CODE Message (Header 0x0D)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **MEDIUM**

The Pass Key Code message, in this case, is used by the console to pass-through RF remote control button presses to the speakers. The Argument byte should contain the key code byte, exactly as received from the RF remote control.

Byte	Name	Description
1	Header	**0x0D**
2	Address	Same as for the console POLL Message.
3	Argument	b7..b0: 8-bit remote control key code.
4	Verifier	XOR of bytes 1 and 2.

PASS KEY CODE Message

9.2.14 INSTALLER SERVER PUSH Message (Header 0x11)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x11**
2	Address	Same as for the console POLL Message.
3	Argument	Argument to be pushed on the argument stack for any subsequent calls of Installer Server Exec. Any call to Installer Server Exec will clear the argument stack.
4	Verifier	XOR of bytes 1 through 3.

INSTALLER SERVER PUSH Message

9.2.15 INSTALLER SERVER EXEC Message (Header 0x12)

Sender: **Console**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description
1	Header	**0x12**
2	Address	Same as for the console POLL Message.
3	Argument	**NOTE: The parameters (arg0, arg1, …) referred to below need to be pushed onto the Installer Server Exec argument stack using the INSTALLER SERVER PUSH command. Before an Installer Server Command is executed it checks that the stack has the correct number of arguments. If the wrong number is present the command will not execute, and the error code will be set to IST_ERR_WRONG_NUM_ARGS. Each command also checks that it is not being run in an invalid command sequence (Ex. Downloading filter coefficients before the filter coefficients have been calculated). If a command is called out of sequence it will not be executed, and the error code will be set to IST_ERR_SEQ_ERROR** 0x00: execute query arg0 = 0x00: Query state arg0 = 0x01: Query last error arg0 = 0x02: Query progress arg0 = 0x03: Query number of frequency bands arg0 = 0x04: Query min number of listening positions arg0 = 0x05: Query max number of listening positions arg0 = 0x06: Query server version id arg0 = 0x10: Query number of speakers in frequency band 0 arg0 = 0x10: Query number of speakers in frequency band 1 … arg0 = 0x1F: Query number of speakers in frequency band 15 arg0 = 0x20-0xFF: undefined 0x01: Initialize server (no arguments) 0x02: Close server connections arg0: SaveFlag (0=Don't Save, 1=Save measurement quality indicators) 0x03: Set gain for voice announcements arg0 : gain (-128 ..+127) in dB 0x04: Perform measurement arg0: id of listening position arg1: id of frequency band arg2: id of speaker inside band arg3: gain (-128 … +127) in dB ..arg4: MuteFlag (0=UnMuted, 1=Mute Excitation) 0x05: Run data test arg0: id of test (see Installer server documentation) arg1: id of listening position arg2: id of frequency band arg3: id of speaker inside band

		0x06: Accept data from last measurement as microphone calibration (no args) 0x07: Stop whatever you are currently doing (no arguments) 0x08: Perform EQ Design arg0: number of listening positions 0x09: Perform Filter Design (no arguments) 0x0A: Upload Filter Coefficients into Equalizer (no arguments) 0x0B- 0xFF: undefined The only actions that can be executed on the Server before it is initialized are 0x00 (query) and 0x01 (initialized). Any call to Exec will clear the argument stack on exit.
4	Verifier	XOR of all preceding bytes, including Header.

INSTALLER SERVER EXEC Message

9.3 Messages Sent by Speakers

The following messages are allowed to be sent by speakers (network slaves**). Note: the address byte for speaker messages is different than for console messages.** Note also that bit 7 of the Address byte is always set to 1 for speaker messages (set to 0 for console messages), to indicate message direction.

9.3.1 POLL REPLY Message (Header 0x80)

Sender: **Speakers**
Total Bytes: **4**
Queue Priority: **LOW**

Byte	Name	Description
1	Header	**0x80**
2	Address	**NOTE: The address byte for speaker messages is different than for console messages:** b3..b0: Same as defined for speaker messages: indicates one of 15 possible rooms, but with 1111b undefined for speakers (since speakers do NOT reply to broadcasts). b7..b4: 0000-1101b: Playing one of 14 possible Zones (0010b=Zone1, 0011b=Zone2, etc.). 1110b: Playing a **local** source. 1111b: **OFF**. Presently playing no local or networked sources.
3	Argument	b7: Mute state: 1=muted. 0=not muted. b6..0: Speaker's Main Attenuation level, in dB. 0000000b = 0dB attenuation (full volume). 0000001b = 1dB attenuation, etc.
4	Verifier	XOR of bytes 1 and 2.

POLL REPLY Message

9.3.2 DOWNLOAD INFORMATION Message (Header 0x8A)

Sender: **Speakers**
Total Bytes: **5-255**
Queue Priority: **MEDIUM**

Byte	Name	Description
1	Header	**0x8A**
2	Address	Same as for speaker POLL REPLY Message (**Different than console messages**).
3	Argument	0x00 = Download Console Type (Data fields TBD) 0x01 = Download Console Software Version (Data fields TBD) 0x02 = Download Console Serial Number (Data fields TBD) 0x03-0x0F = undefined. 0x10 = Download Source Change Data Block (Input Source, Speaker Mode, Effects, Decompressor, and Post Processing bytes follow as Data fields. Length is 10 bytes.) 0x11-0x1F = undefined. 0x20 = Download New Speaker EQ (data contained in this message). 0x21 = Download New Speaker EQ (data contained in S/PDIF stream). 0x22-0x2F = undefined. 0x30 = Download New Application Code (data contained in this message). 0x31 = Download New Application Code (data contained in S/PDIF stream). 0x32-0xFE = undefined. 0xFF = Abort Download.
4	Length	Length of the complete message, in bytes, including the Header, Argument and Checksum fields. 0x00 - 0x04 = illegal. 0x05 - 0xFF = length of entire message, in bytes.
5-254	Data	8-bit data bytes.
Last	Verifier	XOR all previous message fields, including Header.

DOWNLOAD INFORMATION Message

9.3.3 QUERY SPEAKER INFO REPLY Message (Header 0x8C)

Sender: **Speaker**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description	
1	Header	**0x8C**	
2	Address	Same as for speaker POLL REPLY Message (**Different than console messages**).	
3	Argument	Query Argument	Reply Argument(s)
		0x00	Argument: 0x00 = speaker in ON, and able to accept commands. 0x0F = speaker is ON, but unable to accept commands (busy). 0xF0 = OFF, powered-down normally. 0xF1 = OFF, told to power-down fast.
		0x01	Argument: 0x00 - 0x77 (attenuation level) or 0x78 (muted) if speaker is ON, 0xFF if speaker is OFF.
		0x02	Argument 1 = Center level: 0x00-0x0E = 1dB increments below default (down to -15dB). 0x10 = "speaker default." NOTE: default changes with speaker mode. 0x11-0x1E = 1dB increments above default (up to +14dB). Argument 2 = Surround level: 0x00-0x0E = 1dB increments below default (down to -15dB). 0x10 = "speaker default." NOTE: default changes with speaker mode. 0x11-0x1E = 1dB increments above default (up to +14dB).
		0x03	Argument 1 = Overall EQ type: 0x01 = Audio (Normal) EQ Mode. 0x02 = Film EQ Mode. Argument 2 = Treble level: 0x00-0x0E = 1dB increments below default (down to -15dB). 0x10 = "speaker default." NOTE: default changes with EQ type selected. 0x11-0x1E = 1dB increments above default (up to +14dB). Argument 3 = Bass level: 0x00-0x0E = 1dB increments below default (down to -15dB). 0x10 = "speaker default." NOTE: default changes with EQ type selected. 0x11-0x1E = 1dB increments above default (up to +14dB).
		0x04	Argument = Number of speakers active (see Set Speaker Mode message).
		0x05	Argument = explicit input/substream being played (see Set Input message).
		0x06	Argument = Decompressor presently active (see Select Decompressor msg).
		0x07	Argument = Post Processing algorithm active (see Select Post Processing).

		0x08	Arguments 1-3 = 0x000000 - 0xFFFFEF: 3-byte word describing the count of valid bytes received by the speaker via S/PDIF data download. The allowable range for this count is 0x000000 - 0xFFFFEF (up to 16,777,199 bytes), as follows: Argument 1 = valid bytes received count, bits 0..7. Argument 2 = valid bytes received count, bits 8..15. Argument 3 = valid bytes received count, bits 16..23. Argument 4 = received checksum. Arguments 1-3 >= 0xFFFFF0: When arguments 1-3 form a 3-byte word of value greater than 0xFFFFEF, they convey download status information, as follows: Arguments 1-3 = 0xFFFFF0 - 0xFFFFFB: undefined. Arguments 1-3 = 0xFFFFFC: Speaker indicates an Error. Arguments 1-3 = 0xFFFFFD: Speaker is Writing Data. Arguments 1-3 = 0xFFFFFE: Speaker is Reading-In Data. Arguments 1-3 = 0xFFFFFF: Speaker is waiting for Header. Argument 4, in all these cases: 0x00.
		0x09	Argument b0: TRUE if system is installed, i.e. the installer process was run and a valid set of filter coefficients are stored in FLASH. b1: TRUE if installer is currently active. b2: TRUE if system supports installer server (executing the installer process).
		0x0A -0x0F	Undefined.
		0x10	Argument 1 = generic Speaker Type ID: 0x00 = Cobalt II. 0x01 = DigiHiker. 0x02 = LSA2. 0x03 = Ballpark. 0x04 = A2. 0x05 = Knex. 0x06-0xFF = undefined.
		0x11	Arguments 1-6 = ASCII characters describing the speaker's software variant (family subtype).
		0x12	Arguments 1-6 = ASCII characters describing the speaker's software revision, formatted as XXYYYa, respectively. XX=major rev digits (zero-padded), YY=minor rev (zero-padded), and a=a letter or blank.
		0x13	Arguments 1-6 = ASCII characters describing the speaker's manufacturing serial number, if any.
		0x14 -0xEF	Undefined.
		0xFX, where X selects 1 of the 16 possible effects.	Argument b0 - b3 = Status of the effect chosen. 0000b = The effect chosen is not supported in this speaker. 0001b = The effect chosen is supported, but not installed. 0010b = The effect chosen is installed, and presently disabled. 0011b = The effect chosen is installed, and presently enabled. 0100-1111b = undefined. b4 - b7 = 4-bit value identifying the desired effect (1 out of 16). Same as the value of X in the query argument.
4	Verifier	XOR of all previous bytes, including Header.	

QUERY SPEAKER INFO REPLY Message

9.3.4 PASS KEY CODE Message (Header 0x8D)

Sender: **Speakers**
Total Bytes: **4**
Queue Priority: **MEDIUM**

The Pass Key Code message, in this case, is used by speakers to transmit local speaker button presses upstream to the console. The Argument byte should contain the key code byte, exactly as received from the speaker's local IR remote control.

Byte	Name	Description
1	Header	**0x8D**
2	Address	Same as for speaker POLL REPLY Message (**Different than console messages**).
3	Argument	b7..b0: 8-bit remote control key code.
4	Verifier	XOR of bytes 1 and 2.

PASS KEY CODE Message

9.3.5 QUERY INSTALLER SERVER REPLY Message (Header 0x13)

Sender: **Speaker**
Total Bytes: **4**
Queue Priority: **HIGH**

Byte	Name	Description	
1	Header	**0x13**	
2	Address	Same as for speaker POLL REPLY Message (**Different than console messages**).	
3	Argument	Query Argument	Reply Argument(s)
		0x00	Server State: 0x00: not connected 0x01: idle 0x02: opening 0x03: closing 0x04: measuring first trial 0x05: measuring second trial 0x06: designing EQs 0x07: designing Filter Coefficients
		0x01	0x00: no error 0x01-0xFF see Installer Server documentation (InstallerServerDesign.doc)
		0x02	0…100 approximate progress of current action in %, meaningless, if the server is idle.
		0x03	number of frequency bands (2 for Cobalt-II)
		0x04	minimum number of listening positions required to do an adequate EQ design
		0x05	maximum number of listening positions for which the server can provide storage
		0x06	version ID of server
		0x10	number of speakers in frequency band 0 (5 for Cobalt-II).
		0x11	number of speakers in frequency band 1 (1 for Cobalt-II).
		0x12-0x1F	number of speakers in frequency bands 2-15
4	Verifier	XOR of all preceding bytes, including Header.	

QUERY INSTALLER SERVER REPLY Message

10.0 Details Related to Arrow

10.1 Overview

In general, it is desired to define the Arrow Master/Slave transceiver system to be able to plug seamlessly between the FedX console and a networked speaker with no impact to Lifestyle performance. Concerns to be addressed here include:

• Message robustness through Arrow.
The Smart Speaker protocol includes error protection suitable to a wired system only. Arrow must be defined to add extra error protection/correction to the Smart Speaker messages as appropriate to its channel error profile.

• Message latency through Arrow.
The FedX console polling structure has been defined to require a fixed, predictable response time from speakers. A technique needs to be developed to reconcile Arrow's delays with this requirement.

• Handling of Retries
The FedX console polling cycle can be held off to accommodate immediate message retries for hardwired speakers. Any Arrow solution needs to include similar support for retries, when required by the speakers.

10.2 Message Robustness Through Arrow

Arrow agrees to take responsibility for the integrity of messages send through its wireless channel. This applies to messages sent downstream from the console to speakers (Arrow refers to this as the "downlink") as well as messages sent back in return (the "backlink"). Any required extra error protection/correction will be added by the transceiver to guarantee an agreed-upon level of robustness.

10.3 Managing Arrow Message Latency

Presently, Arrow downlink messages are expected to have about 20mSec of latency reaching the desired speaker (with high robustness), and return messages have an opportunity to be sent once every 20mSec (with somewhat less robustness). Since Smart Speaker network throughput would be drastically diminished if the system pended on replies long enough to tolerate such delays, a scheme is suggested to work around them.

The foundation of this scheme is that **polls from the console will NOT be sent across the Arrow link**. Instead, Arrow Slaves will locally poll the speakers attached to them and transmit replies to the Master once every 20mSec. The Master will buffer these replies and locally pass them on to the console in response to appropriate polls. Details are provided below. Default replies (Header 0x80) will also be generated by the Master in response to all other (i.e., higher-priority) messages from the console to speakers, in some cases until a query reply is returned via the backlink.

10.4 General Responsibilities of the Slave

In general, the Slave needs to implement a Smart Speaker interface which appears to the speaker to be "identical-enough" to being connected to the FedX console. This requires polling the speaker in a manner similar to the console, interrupting this polling to immediately pass downstream messages through to the speaker, and assembling all poll reply information and transmitting it to the console via the backlink. The Slave's polling will be simplified somewhat, given the topological advantage that one Arrow Slave will be dedicated to every networked speaker.

10.4.1 The Slave's Local Polling Cycle

Since polls generated by the console are never sent to Slaves, each Slave must autonomously identify its associated speaker and manage a local polling cycle. Arrow Slaves will **only need to poll the single speaker attached to it--**

no local polling will be required for other speakers. To determine the Room Code of the speaker attached to it, the Arrow Slave shall institute a simpler-version of the polling cycle implemented in the Postman2/FedX console, as follows:
1. The Arrow Slave will continuously poll all possible speakers until it finds one which replies.
2. Subsequently, the Arrow Slave will poll ONLY that speaker, at a 20mSec rate.
3. If the speaker stops replying, the Arrow will begin polling all possible speakers again, until a replying speaker is found (could be a new Room Code).

To ensure that one poll response is assembled in time for each 20mSec backlink time period, the Arrow Slave's local polling period will be **time-synchronized by the Slave such that poll exchanges occur immediately before each backlink message is sent** (beginning about 6mSec earlier, allowing time to assemble the reply for inclusion in the backlink message). Speaker high-priority messages (local button presses, etc.) will therefore suffer up to about 20mSec of latency before having an opportunity to be transmitted (the first time) from a Slave to the Master.

10.4.2 Priority of Console Messages Over Local Polling

As previously outlined, delivering console messages is a higher priority than retrieving polling information from a speaker. Therefore, when a Slave receives a message from the console intended for the speaker attached to it (a PASS_KEY_CODE message, for example), the message should be passed-through to the speaker **immediately**, regardless of the timing of its local polling cycle.

10.4.3 Local Retry Rules for the Slave

Each Slave needs to manage two situations where retries might be required:
- The case where a message passed-through to a Slave's local speaker fails to be replied-to by the speaker.
- The case where a message sent by the Slave (from a speaker) to the console via the backlink fails to be properly received by the Master (a PASS_KEY_CODE message from the speaker, for example).

In the first case, **the Slave must locally manage an immediate retry of the console's message to the speaker.** Only one retry will be defined to be necessary. Again, it is acceptable if this retry interferes with a scheduled local poll for the speaker.

The second case presumes some form of feedback from the Master to the Slave immediately following an attempted message transfer via the backlink. **Such feedback should be defined to be included during the very next Arrow 20mSec polling cycle.** Effectively, this should take the form of a Request for Retry from the Master to each effected Slave. If a Slave sees this Request for Retry, then on its next opportunity to send information via the backlink it should once again forgo sending polling information to the Master in favor of re-sending the last speaker message. This retry mechanism could continue as long as necessary to get the message through to the Master, and should be governed by the Master. 100mSec (5 retries) might be a good limit here. Note that speaker polling can continue through the retry situation, allowing speaker polling information to be refreshed (**overwritten, NOT buffered**) and messages to be buffered in a (small) queue awaiting the next opportunity to transmit via the backlink.

10.5 General Responsibilities of the Master

The Arrow Master, as described, must satisfy the FedX console polling protocol by replying in the proper time with all messages and polling information received from the speakers/Slaves; in effect, "fooling" the console into thinking that it is hardwired to its set of speakers. A few specific duties fall out of this, detailed below.

10.5.1 Buffering Speaker Poll Replies to Use Locally

The Arrow Master must immediately send a Poll_Reply (0x80) Message in response to **all console messages**, for ALL connected speakers, **whenever no message of higher priority is pending from a speaker**. To generate these Poll_Replies, it must buffer the backlink messages resulting from Slaves' local polling cycles and use this information to format proper replies within the required timing constraints.

Since polling information represents an ongoing update of the same state variables, **poll status buffers for each speaker will be only one message deep, and will be continually refreshed as more recent data is retrieved from speakers.** If a Master receives invalid poll reply data from a given speaker (invalid checksum, etc.), the Master will use the (older) data stored in its buffer to reply to the console. If the Master fails to receive valid poll reply information from a given speaker for 5 Seconds, the Master will empty the poll status buffer associated with that speaker and begin withholding replies when polled by the console. It will be the console's responsibility to remove that speaker from its list of ON speakers, etc.

10.5.2 Substituting Higher-Priority Speaker Messages for Poll Replies

As described in previous sections, speakers must reply to console messages with the highest-priority return message pending (where Poll Replies are used as a default). Therefore, to simulate this, the Arrow Master must identify when a message of higher-priority than a Poll_Reply is pending from a given speaker, and substitute this message for the default Poll_Reply after the next console message for that speaker. If both a medium-priority and a high priority message are pending, the high priority message should be sent first.

Passing these higher-priority messages through to the console should **not** corrupt the poll status buffers for each speaker. Once the higher-priority message has been passed to the console, the Master should return to replying with Poll_Reply messages.

10.5.3 Responsibilities for Higher-Priority Console Messages

As mentioned, the Arrow Master must reply to **all** console messages. This is true, regardless of the priority of the console message (see message definitions). Replies must be started within the 767uSec window described previously. Once replying to the console, the Arrow subsystem assumes responsibility **for delivering all medium or higher-priority console messages reliably to the speaker (including managing retries),** since the console hereafter believes them to have been received by the speaker and may lose the ability to manage retries at a higher level.

10.5.4 Responsibilities During a Query Exchange

As described in previous sections, when a FedX console sends a query to a specific speaker, it (the console) **will stop polling other speakers until the reply is returned. It will, instead, continuously poll the speaker for which the reply is pending.** In this situation, the Arrow Master simply continues to reply to the console's polls until either:

1. The console decides to give up and resume polling other speakers, or
2. The query reply comes through from the desired speaker. The Master shall give this a higher priority than poll replies, and therefore transfer it to the console in response to the next available poll.

During these query exchanges, the FedX system is most sensitive to the Arrow backlink delay (no work-arounds are in place). Since FedX/Hermes systems may require more frequent speaker queries than Postman1 systems, it is highly desirable for Arrow to find a way to reduce backlink delays to the point where reliable replies are available within 50mSec of being sent by the speaker. This may require Arrow to forego collecting channel status information temporarily when a high-priority backlink message needs to be sent.

10.5.5 Requesting Retries from Slaves

Since the Arrow backlink may be less reliable than the downlink, it is possible that high-priority messages from a Slave may arrive corrupted at the Master. In such situations, it is required that the Master continually request retries from the Slave until the message is transferred intact. See Section 10.4.3.

11.0 Details Related to Cobalt2

Cobalt 2 presently speaks the 4800 baud Normal protocol. It might be an advantage overall to upgrade it to support the 19.2kbps variant outlined in this document. The minimum required changes would be:

- Increase in bit rate to 19.2kbps.
- Decrease reply latency to support the 767uSec maximum. NOTE: query replies can still take longer for Cobalt2 to generate, as long as it continuously generates (the default) Poll Replies while assembling the query reply.
- Add support for the new poll.
- Eliminate 1-byte ACK's for all messages, in favor of 4-byte poll replies, which are implicit ACK's.
- Adhere to the new exchange procedure for queries/replies.

12.0 Details Related to Ballpark

Ballpark is required to operate in two configurations: one as a stand-alone product with perhaps an accessory pedestal unit, and the second as a non main-room Smart Speaker attached to a Lifestyle console (with, by definition, NO pedestal unit attached). It is possible that the Smart Speaker bus could be used for communication between the Ballpark unit and its pedestal, in the first configuration. The following explores how.

12.1 Network Master/Slave Responsibility

Using the polling method described in previous sections, the P2/FedX Smart Speaker protocol could enable the sort of bi-directional data required between Ballpark and its pedestal without contention/collisions. A network Master must be defined, however, to manage polling.

Since the Ballpark unit will already have code allowing it to be a Smart Speaker slave, one possibility would be for polling to be managed by the pedestal unit (essentially acting as a console, or master). This would have the minimum impact on Ballpark. If Ballpark ever needs to attach to more than one pedestal simultaneously (a hard drive AND an satellite radio receiver, for example), this might be problematic. However, if this is a non-requirement, a simplified polling cycle like that used by the Arrow Slaves could easily be defined to be implemented into the pedestal.

12.2 Latency of Messages

If the pedestal implements a simplified polling cycle like Arrow's (polling only a single room, known to be the room code used as a default by the Ballpark unit, for example), information sent from the pedestal to Ballpark would have essentially no delay (sent immediately when required). This would be an advantage when sending MP3 playback time-ticks (including FFWD/FREV modes) and TTAG updates originating from the hard drive and intended for the Ballpark display.

Messages from Ballpark to its pedestal would suffer one polling cycle's worth of delay. This would seem acceptable for transport commands from Ballpark to the hard drive, for example, if kept to well below 50mSec (the 20mSec polling cycle defined for Arrow Slaves might work well).

12.3 Sharing Smart Speaker with ETAP

ETAP messages could potentially be substituted for normal Smart Speaker commands if Ballpark remained a slave. In this case, the PC could generate control commands/queries whenever it needed to, and could retrieve replies from Ballpark by generating polls until a reply returned. Workarounds like those in the standard ETAP protocol would be needed if the PC could not respect the packetization requirements of Smart Speaker messages (no gaps between message bytes). In the limit, these workarounds would require the Ballpark to default to ETAP mode (with a PC-centric set of message timing rules) until it received a Smart Speaker poll.

13.0 Details Related to SA2/SA3

SA2/3 is required to support the ability to be ganged together with other SA2/3's—that is, to be connected together in such a way as to allow all speakers to respond, as a group, to the same RF remote control messages (turn on/off, raise and lower volume, etc., together).

To achieve this, SA2/3 will provide a speaker output mini-DIN connector next to its input connector, for the purposes of daisy-chaining other SA2/3's. The Smart Speaker bus connected to this output mini-DIN connector will be the same as brought in the input mini-DIN (tied directly together). A DIP switch on the SA2/SA3 will determine whether the unit listens and replies normally on the bus (Standard Mode), or simply listens to commands (as well as acting on them) without replying (Ganged Mode). It is this ability in Ganged Mode to listen/respond to bus commands while suppressing bus replies which makes it possible to gang SA's together without corrupting network communications.

13.1 Standard Mode

In Standard Mode (selected by a DIP switch), the AS2/3 will listen and respond to all properly-addressed console Smart Speaker messages as defined earlier in this spec. This includes generating all specified message replies within the defined time limits. The full set of messages supported includes:

13.1.1 Pass_Key_Code

Arguments supported: ON/OFF, Volume_Up, Volume_Down, and Mute Toggle.

13.1.2 Mute_All_Assert

13.1.3 Mute_All_De-assert

13.2 Ganged Mode

In Ganged Mode (selected by the same DIP switch), the SA2/SA3 will never generate smart speaker replies of any sort—it will behave as a listen-only speaker. In this mode, it will also ignore all smart speaker commands, including those addressed to it. Instead, it will listen for Poll replies from a Standard Mode speaker sharing its room address, and use these replies to control its state. If no Standard Mode speaker exists on the bus with the same room address, a Ganged Mode SA2/SA3 will essentially become inoperable.

13.2.1 Maintaining State Synchronization

It is desired that all SA2/SA3's sharing the same room address cooperate to stay in synchronization, regarding volume level, mute status and on/off status. This is achieved as follows:

Only one SA2/LSA3 with a given room address may exist on the Smart Speaker network in Standard Mode at a given time. All other SA2/SA3's sharing the network with the same room address MUST be in Ganged Mode. The Poll Replies from the Standard Mode speaker will contain its on/off status, mute status and volume level (see message definitions earlier in this document). All SA2/SA3's in Ganged Mode shall ignore all speaker commands; instead, they must listen for Poll Replies from a Standard Mode speaker sharing their room address, and adjust their own state to match the state described in those Poll replies.

13.3 Legacy Support

SA2 and SA3 only need to respond to 6 Legacy Mode smart speaker messages: ON/Unmute, Off, Volume Up, Volume Down, and Mute Toggle. Reference the CD-5 Serial Interface spec OP173549.