PFx Brick

Host Interface Control Document

for

USB and Bluetooth LE

Document No.: 11 80 17 10001

Rev. 3.38

Aug 1, 2021

Ottawa, ON CANADA

www.fxbricks.com

Contents

Revision Notes

Changes made to each version of this document are summarized in the table below.

Rev	Change Notes
2.1	The PFX_CMD_GET_ICD_REV message was added so that firmware can report which revision of ICD it conforms with. The COMMAND_IR_LOCKOUT_TOGGLE command was added to the COMMAND byte of event/action definition. The MOTOR_ACTION_ID definitions for MOTOR_STOP and MOTOR_COAST were redefined to MOTOR_ESTOP and MOTOR_STOP respectively. New PFX settings bit added to the PFx Brick configuration called Audio DRC. The PFX_CMD_WRITE_SN and PFX_CMD_READ_SN messages were added to manage PFx Brick serial number assignment.
2.2	Added description for the traffic light combo light f/x.
2.3	Added new message PFX_CMD_GET_CURRENT_STATE to report internal operating state of motors, lights, audio, etc.
2.4	Modified the format of the PFX_CMD_GET_CURRENT_STATE message to report motor PWM speed. Corrected the numeric definitions of BAR_STYLE used with the sound bar light f/x.
2.5	Added new message PFX_CMD_GET_IRRX_STATUS message to report low level data from the IR receiver processor.
2.6	Added new message PFX_CMD_SET_AUDIO_EQ message to set audio equalization levels. The valid range for bass/treble EQ values has been set to -20 to +20 dB in the configuration. Modified the PFX_CMD_GET_CURRENT_STATE message format to send the internal millisecond counter data.
2.7	Added new parameter SWEEP_STYLE for the COMBOFX_LINEAR_SWEEP and COMBOFX_BARGRAPH_SWEEP combination light f/x.
2.8	Added new MOTOR_STEP parameter for Lego compatible 7 step operation. Added new COMBOFX_LAVA_LAMP combo light f/x Added new WHELEN_STYLE parameter for a random program of flashing sequences.
2.9	Added new motor configuration bit "TLG Mode" to add emulation of the Lego IR receiver motor control.

Rev	Change Notes
3.0	Revised the PFX_CMD_GET_STATUS message to include comprehensive product identification. This includes a new fields for USB PID, Product Number, Product Descriptor, a new 4-byte Serial Number, and a new 2-byte Firmware Version. Deprecated the Product ID, Hardware Version, Firmware Version, and Serial Number fields in the the PFX_CMD_GET_CONFIG message. Revised the PFX_CMD_WRITE_SN and PFX_CMD_READ_SN messages to accommodate the new 4-byte serial number format. Added new COMMAND bytes to the event/action LUT. Renamed EVT_DEFAULT_EVENT to EVT_STARTUP_EVENT1 and added 3 more startup events.
3.1	Revised the PFX_CMD_GET_ICD_REV message to support a 2-byte revision numbering scheme.
3.11	Added a new RETRIGGER parameter to the SOUNDFX_PLAY_ONCE sound f/x. Added new COMBOFX_LASER_CANNON combination light f/x. Corrected the description of EVT_STARTUP_EVENTx for both the PFX_CMD_GET_EVENT_ACTION, PFX_CMD_SET_EVENT_ACTION messages.
3.12	Changed the format of the PFX_CMD_GET_AUDIO_LUT_ENTRY message to also return the start address of the audio sample data. The File Size field now represents the Data Size of audio sample data, not the total file size. These changes reflect internal changes in the firmware to be tolerant of different WAV file formats including LIST and INFO chunks. Changed the format of the PFX_CMD_ADD_AUDIO_DATA message to report progress information for lengthy flash erase operations which result in a PFX_ERR_TRANSFER_BUSY_WAIT status response code.
3.13	Added suggested default values for all of the light f/x.
3.14	Added new LIGHTFX_BROKEN_LIGHT single light f/x. Added new LIGHTFX_STATUS_INDICATOR single light f/x. Revised the definition of the light output 7 for emergency flashers from solid to 2x flasher. Added a Silent flag for the PFX_CMD_GET_ICD_REV message so that it can be used for covert connection monitoring.
3.15	Added new LIGHTFX_SOUND_MODULATED single light f/x. Added new FLICKER_ON parameter to LIGHTFX_ON_OFF_TOGGLE light f/x. Added new definition to LIGHT_PARAM4 for single light f/x to assert output state beyond simple toggle on/off. Increased the audio LUT size from 16 to 32 entries.
3.16	Added new LIGHTFX_MOTOR_MODULATED single light f/x. Corrected the parameter definition for LIGHTFX_SCIFI_ENGINE_GLOW

Rev	Change Notes
3.17	Added new COMMAND_RESTART command Added additional response data to the PFX_CMD_GET_CURRENT_STATE command Changed references of PFXBrick to PFx Brick to match product naming conventions
3.20	Updated the memory map to show the addition a File Allocation Table file system and removal of the Audio LUT structure. Added a description of a newly introduced file system applied to the flash memory. New USB command messages have been added to interact with the file system Deprecated the following messages associated with audio file access: PFX_CMD_ADD_AUDIO_FILE, PFX_CMD_ADD_AUDIO_DATA, PFX_CMD_ADD_AUDIO_DONE, PFX_CMD_GET_AUDIO_FILE, PFX_CMD_GET_AUDIO_DATA, PFX_CMD_ERASE_AUDIO_LUT Added an error code reference for file system access commands Deprecated the PFX_CMD_DIAG_LED command Changed the command byte value of PFX_CMD_GET_ICD_REV to 0x08 from 0x00 since 0x00 seems to be a reserved report byte usage value for USB HID report packets Changed the format of the configuration data in PFX_CMD_SET_CONFIG and PFX_CMD_GET_CONFIG to support optional individual brightness adjustments for each lighting channel. Changed the format of the PFX_CMD_GET_STATUS message to include new fields for USB VID and firmware build no.
3.21	Changed the PFX_CMD_FILE_FORMAT_FS message to specify different formatting modes. Added a new request to the PFX_CMD_FILE_DIR command. Changed the PFX_CMD_GET_CURRENT_STATE message to add more status parameters.
3.22	Changed the PFX_CMD_FILE_GET_FS_STATE message to report both free and empty sectors.
3.23	Added an INVERT parameter to LIGHTFX_SOUND_MODULATED single light effect.
3.24	Changed product ID and corresponding descriptors. Added product ID reference table as an appendix. Added new motor actions MOTOR_SET_SPD_TIMED, MOTOR_OSCILLATE, MOTOR_OSCILLATE_BIDIR, MOTOR_OSCILLATE_BIDIR_WAIT, MOTOR_RANDOM, MOTOR_RANDOM_BIDIR, MOTOR_SOUND_MODULATED Added new motor parameters DURATION and MOTOR_PERIOD Extended the definition of the MOTOR_SPEED parameter to allow for higher resolution set speed. Added support for additional IR remote controls: LEGO® RC Train remote, Sparkfun COM-11759 mini IR remote, and Adafruit 389 mini IR remote. These definitions expand the Event/Action LUT.

Rev	Change Notes
3.25	Added new Bluetooth communications commands PFX_CMD_GET_BT_STATUS, PFX_CMD_SET_BT_POWER, PFX_CMD_SEND_BT_UART, PFX_CMD_RECEIVE_BT_UART
3.30	Changed the format of the configuration data in PFX_CMD_SET_CONFIG and PFX_CMD_GET_CONFIG to support new parameters. Specification of the user-defined name has been moved from the configuration messages to two new messages: PFX_CMD_SET_NAME and PFX_CMD_GET_NAME Added new section discussing the Bluetooth interface services and message format Added an introduction to this document to reinforce the commonality of both USB and BLE interfaces for remote configuration and control
3.31	Added a new command PFX_CMD_SEND_EVENT to simulate remote control events over USB or BLE. Expanded the definition of the MOTOR_PERIOD parameter to specify both an ON and OFF duration.
3.32	Added the notificaiton mechanism to allow USB and BLE connected hosts to subscribe to notifications from the PFx Brick. This adds the PFX_CMD_SET_NOTIFICATIONS and PFX_MSG_NOTIFICATION commands to the host control interface.
3.33	Added new SOUND_FX_ID: SOUND_FX_PLAY_IDX_MOTOR for realistic motor/prime mover sound effects based on sampled sound files indexed by changes in motor speed. Added new new SOUND_FX_ID: SOUND_FX_PLAY_RAND to randomly playback a specified sound file continuously. Changed the format of the configuration data in PFX_CMD_SET_CONFIG and PFX_CMD_GET_CONFIG to store speed boundaries between indexed motor speed sounds. Added new TRAFFIC_STYLE type "European 2".
3.34	Added new TRAFFIC_STYLE type "European 2 with pedestrian crossing" Changed the format of the PFX_CMD_GET_CURRENT_STATE return message Added new items to the SOURCE1 parameter Deprecated the namespace prefix of PFX_USB_CMD_ and replaced it with the more appropriate PFX_CMD_ prefix. All references to either namespace are considered synonymous. Changed the format of the PFX_CMD_FILE_DIR response message to include the request type in the response to simplify parsing by the host.
3.35	Deprecated the PFX_CMD_GET_AUDIO_LUT_ENTRY, PFX_CMD_GET_AUDIO_CAPACITY messages Added new PFX_CMD_FILE_DIR request type PFX_DIR_REQ_SET_ATTR_MASKED_ID Expanded the definition of the file "User Attributes" field to tag files for use with indexed motor sound samples Added new file attribute for PFX file extensions (representing scripts)

Rev	Change Notes
3.36	Changed the SOUNDFX_STOP definition to stop audio playback of the file specified in SOUND_FILE_ID rather than all audio playback. Deprecated the PFX_CMD_GET_LAST_IR_MSG, PFX_CMD_VERIFY_CONFIG, PFX_CMD_VERIFY_EVENT_LUT messages Added new error code PFX_ERR_TRAP_BROWNOUT_RST Corrected the PFX_CMD_READ_I2C, PFX_CMD_WRITE_I2C messages description to conform with actual firmware implementation. Changed the USB PID to the officially sublicensed PID from Microchip for the PFx Brick.
3.37	Removed the 2x auxiliary flasher (on light ch. 7) for EVT_COMBOFX_EMCY_TWSONIC, EVT_COMBOFX_EMCY_WHELEN combo light effects Added TRANSITION parameter as LIGHT_PARAM5 to the EVT_COMBOFX_ALT_FLASH combo light effect to specify behaviour when toggling effect Added EVT_COMBOFX_DRAGSTER combo light effect Added EVT_COMBOFX_FORMULA1 combo light effect Added EVT_COMBOFX_RUNWAY combo light effect Rename TRAFFIC_STYLE ”European 2” to ”International” Added new TRAFFIC_STYLE ”International 2” Added new MOTOR_SET_SERVO to MOTOR_ACTION_ID to set servo motor position Added new MOTOR_POS motor parameter to specify servo motor position Added new step size for servo motor increments in the MOTOR_STEP parameter Added new script language support for the PFx Brick Added new COMMAND_RUN_SCRIPT to the COMMAND byte of event/action. Added new PFX_CMD_RUN_SCRIPT message to execute a script file Added a new User Attribute for file system to mark text files for use with scripting Added new error codes Changed the format of the PFX_CMD_GET_CURRENT_STATE message response Added 3 new request types for the PFX_CMD_FILE_DIR command message: PFX_DIR_REQ_GET_NAMED_FILE_ID, PFX_DIR_REQ_GET_SMALL_DIR_ID, PFX_DIR_REQ_GET_SMALL_DIR_IDX Re-organized the document by putting reference descriptions of the memory map, file system, etc. at the end

Rev	Change Notes
3.38	Added new file attributes for multiple gated playback sound files. Added reserved file IDs as an alternative mechanism for marking special files for indexed playback Added new `EVT_BUTTON_PRESS`, `EVT_BUTTON_LONGPRESS`, `EVT_BUTTON_DOWN`, and `EVT_BUTTON_UP` events which are triggered by an attached touchLAB accessory. Added new `EVT_BLE_CONNECT`, `EVT_BLE_DISCONNECT`, `EVT_USB_CONNECT`, and `EVT_USB_DISCONNECT` events. Added new scripting language keywords `set`, `event`. Added support for variables, nested loops, and wait events for pushbuttons, setting of configuration values and event actions. Added new sound events `EVT_SOUND_FILE_SEEK` and `EVT_SOUND_FILE_SCRUB` Added new request type for `PFX_CMD_FILE_DIR` command message: `PFX_DIR_REQ_CHANGE_FILE_ID` Added new configuration values: `Rapid Accel Thr`, `Rapid Decel Thr`, `Brake Rate Thr`, `Brake Speed Thr` for use with indexed playback sound schemes. Corresponding changes to `PFX_CMD_GET_CONFIG` and `PFX_CMD_SET_CONFIG` Changed data returned from `PFX_CMD_GET_CURRENT_STATE` Added new definitions for `SOURCE2` light parameter: 0x40 = Button State, 0x80 = Gated Motor Playback Trigger Added special script filename `startup.pfx` which auto executes after power on or reset.

1 Introduction

The PFx Brick injects new possibilites of animation and control for LEGO® models by offering rich capabilities for controlling Power Functions motors, diverse lighting effects and for the first time, user defined sound effects. These features have a wide range of operational possibilies and characteristics. In order to use and configure these features to a user's desired application, a host computer or mobile device uses a software application to make this process simple and efficient. Fx Bricks offers the PFx App to perform this function; however, it is possible for any 3rd party to make a software application to interact with the PFx Brick as well.

In order for a host application to interact with the PFx Brick, it must connect to the PFx Brick via either the standard USB interface or optionally with a Bluetooth Low Energy (BLE) connection. Both of these physical interfaces offer a common command and control message facility described in this Interface Control Document (ICD).

2 PFx Brick USB HID Device Class

The PFx Brick firmware includes a USB HID compliant interface device for communications with USB attached hosts. This will allow host applications to configure and update any attached PFx Brick without the need for custom device drivers. An attached PFx Brick should automatically trigger the host operating system to enumerate the PFx Brick within the USB stack and recognize it as a USB HID compliant device with custom endpoints.

2.1 PFx Brick Vendor and Product ID (VID/PID)

The PFx Brick unoffical vendor ID is 0x04D8 (Microchip Inc.'s registered VID) The PFx Brick USB product ID is 0xEF74 (Microchip vendor sublicensed PID for the PFx Brick)

To find the PFx Brick using the HID API, the following code could be used:

```
device = hid_open(0x04D8, 0xEF74, NULL);
```

2.2 Message Packet Format

USB HID message packets are exchanged via two buffers:

1. OUT endpoint 64 bytes (data from the host)
2. IN endpoint 64 bytes (data to the host)

The PFx Brick will respond to commands issued by the host using a set of customized command messages. The format of these message packets is described in this document. These messages facilitate a wide range of functionality and will continue to evolve over the lifecycle of the PFx Brick.

3 Bluetooth Low Energy

Certain PFx Brick models are fitted with a Bluetooth Low Energy (BLE) v.4.2 compliant interface. This interface allows connected BLE hosts to control and interact with the PFx Brick identically to a USB connected host. The messages described in this document are identically formatted for transport via USB and/or BLE.

The BLE interface on the PFx Brick is configured to operate as a "transparent UART". That is, it provides the same functionality as a bi-directional asychronous serial interface. The PFx Brick advertises this as a BLE compliant GATT service with characteristics assigned to transmit and receive operations. Additionally, the PFx Brick also offers the standardized Bluetooth Device Information service GATT for detailed identification of the PFx Brick.

The BLE GATT services which the PFx Brick advertises are as follows:

Service UUID Service	`0x180A` Device Information	
	Characteristic UUID Characteristic Descriptor	`0x2A29` Manufacturer Name String
	Characteristic UUID Characteristic Descriptor	`0x2A24` Model Number String
	Characteristic UUID Characteristic Descriptor	`0x2A25` Serial Number String
	Characteristic UUID Characteristic Descriptor	`0x2A27` Hardware Revision String
	Characteristic UUID Characteristic Descriptor	`0x2A26` Firmware Revision String
	Characteristic UUID Characteristic Descriptor	`0x2A28` Software Revision String
	Characteristic UUID Characteristic Descriptor	`0x2A23` System ID
	Characteristic UUID Characteristic Descriptor	`0x2A2A` IEEE Regulatory Certification

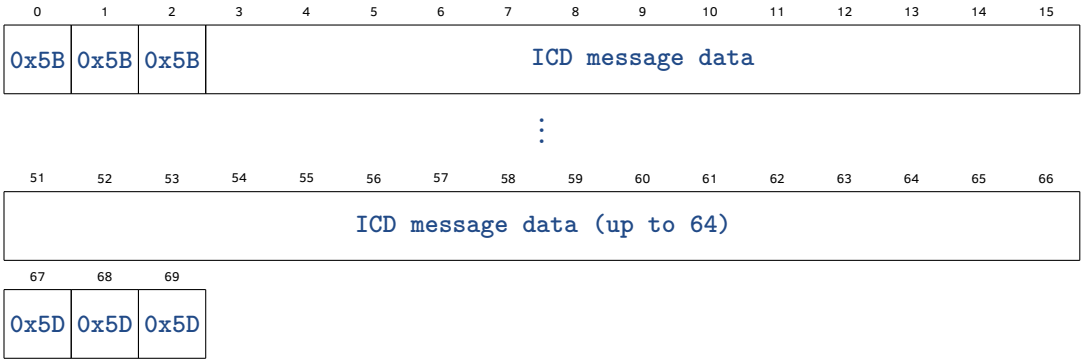
Service UUID	49535343-FE7D-4AE5-8FA9-9FAFD205E455	
Service	Transparent UART	
	Characteristic UUID	49535343-1E4D-4BD9-BA61-23C647249616
	Characteristic Descriptor	UART Receive
	Characteristic Properties	Write Without Response Write Notify Indicate
	Characteristic UUID	49535343-8841-43F4-A8D4-ECBE34729BB3
	Characteristic Descriptor	UART Transmit
	Characteristic Properties	Write Without Response Write
	Characteristic UUID	49535343-A4C8-39B3-2F49-511CFF073B7E
	Characteristic Descriptor	UART Transmit (with response)
	Characteristic Properties	Write Notify

The PFx Brick normally advertises its presence periodically so that it can be discovered by a connecting host. Once discovered, a host can connect to the PFx Brick and ask for service descriptors for both the Device Information and Transparent UART. It can then send and receive ICD messages with the Transparent UART service by using the UART Receive and Transmit characteristics.

3.1 Message Packet Format

BLE message packets are exchanged via two buffers which are part of the UART Transmit and Receive charactersitics. Internally, these buffers are limited to 20 bytes each. Therefore, the standard 64 byte ICD messages will be broken up into an integral number of 20 byte transactions to perform the transfer. From the point of view of the PFx Brick, this process is transparent. However, for the connecting host, extra processing will be required to assemble/disassemble ICD messages into 20 byte payloads.

Messages are sent to the PFx Brick via the `UART Transmit` service characteristic. The format of the message block is as follows:

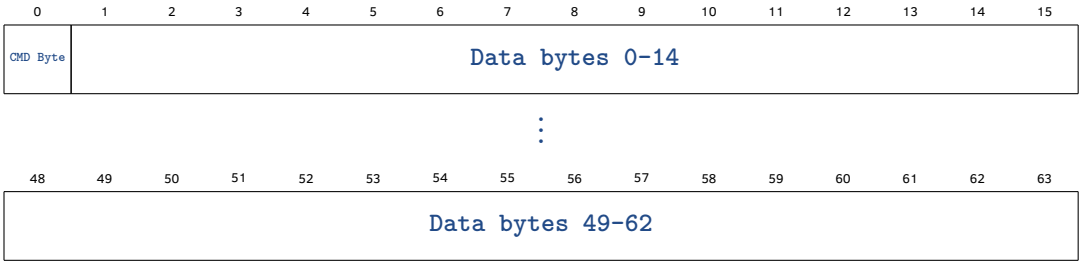


Note that all messages sent to the PFx Brick are pre-delimited with 3x `"["` characters (91 decimal, 0x5B hex) and post-delimited with 3x `"]"` characters (93 decimal, 0x5D hex).

The PFx Brick always sends a response to every transmitted message it receives. These responses are sent as raw data bytes without any pre or post delimiters in exactly the same format as they would be for USB connected hosts.

4 Host Command Messages

The USB HID class supports the exchange of message buffers between the host and a device of up to 64 bytes. The PFx Brick message definition consists of various command messages which originate from the host. The structure of these messages is as follows:



The CMD byte is a numeric literal which specifies the command. A command message may have up to 63 additional data bytes associated with it depending on its purpose. A description of each command is given below along with the format of a device responses if applicable. The device response will prefix its response in byte 0 with the CMD byte xor-ed with 0x80, i.e. it will send the command byte back with the MSB set to '1'.

The following tables show the host CMD bytes grouped by functional category. Also shown is the applicability and/or support of each message within the different software operational contexts. For example, the bootloader application context will not have support for every message since it has limited resources to for processing.

Operation and Configuration Commands

CMD	Nmemonic	Context		
		Firmware	Bootloader	Host App
0x08	PFX_CMD_GET_ICD_REV	y	y	y
0x01	PFX_CMD_GET_STATUS	y	y	y
0x02	PFX_CMD_SET_FACTORY_DEFAULTS	y		y
0x03	PFX_CMD_GET_CONFIG	y		y
0x04	PFX_CMD_SET_CONFIG	y		y
0x06	PFX_CMD_GET_CURRENT_STATE	y		y
0x07	PFX_CMD_GET_NAME	y		y
0x09	PFX_CMD_SET_NAME	y		y

Event/Action LUT Commands

CMD	Nmemonic	Context		
		Firmware	Bootloader	Host App
0x11	PFX_CMD_GET_EVENT_ACTION	y		y
0x12	PFX_CMD_SET_EVENT_ACTION	y		y
0x13	PFX_CMD_TEST_ACTION	y		y
0x15	PFX_CMD_SEND_EVENT	y		y

Audio Commands

CMD	Nmemonic	Context		
		Firmware	Bootloader	Host App
0x20	PFX_CMD_INC_VOLUME	y		y
0x21	PFX_CMD_DEC_VOLUME	y		y
0x2A	PFX_CMD_SET_AUDIO_EQ	y		y

Service Commands

CMD	Nmemonic	Context		
		Firmware	Bootloader	Host App
0x30	PFX_CMD_LOAD_FIRMWARE_FILE	y	y	y
0x31	PFX_CMD_LOAD_FIRMWARE_DATA	y	y	y
0x32	PFX_CMD_LOAD_FIRMWARE_DONE	y	y	y
0x34	PFX_CMD_READ_BOOTCONFIG		y	
0x37	PFX_CMD_REBOOT	y	y	y

File System Access Commands

CMD	Nmemonic	Context		
		Firmware	Bootloader	Host App
0x40	PFX_CMD_FILE_OPEN	y		y
0x41	PFX_CMD_FILE_CLOSE	y		y
0x42	PFX_CMD_FILE_READ	y		y
0x43	PFX_CMD_FILE_WRITE	y		y
0x44	PFX_CMD_FILE_SEEK	y		y
0x45	PFX_CMD_FILE_DIR	y		y
0x46	PFX_CMD_FILE_REMOVE	y		y
0x47	PFX_CMD_FILE_FORMAT_FS	y		y
0x48	PFX_CMD_FILE_GET_FS_STATE	y		y
0x4B	PFX_CMD_RUN_SCRIPT	y		y

Bluetooth Interface Commands

CMD	Nmemonic	Context		
		Firmware	Bootloader	Host App
0x50	PFX_CMD_GET_BT_STATUS	y		
0x51	PFX_CMD_SET_BT_POWER	y		
0x52	PFX_CMD_SEND_BT_UART	y		
0x53	PFX_CMD_RECEIVE_BT_UART	y		

Notification Commands

CMD	Nmemonic	Context		
		Firmware	Bootloader	Host App
0x60	PFX_CMD_SET_NOTIFICATIONS	y		y
0x61	PFX_MSG_NOTIFICATION	y		y

Low Level Test/Debug Commands

CMD	Nmemonic	Context		
		Firmware	Bootloader	Host App
0x70	PFX_CMD_STATUS_LED	y		
0x72	PFX_CMD_WRITE_SPI	y		
0x73	PFX_CMD_READ_SPI	y		
0x74	PFX_CMD_WRITE_I2C	y		
0x75	PFX_CMD_READ_I2C	y		
0x76	PFX_CMD_READ_FLASH	y		
0x77	PFX_CMD_GET_IRRX_STATUS	y		

4.1 PFX_CMD_GET_ICD_REV

This command queries the revision number of the Interface Control Document/Specification (ICD) that the PFx Brick supports. The returned version number will correspond to the revision number of this document. This will give both firmware and host software development a common reference point for determining compatibility. The ICD revision number is independent of both the firmware revision and host software revision/build state. It is possible that several consecutive versions of firmware may support a common revision of ICD.

Host command packet:

0	1	2	3	4
0x08	0x60	0x0D	0x01	Silent

Device response packet:

0	1	2
0x88	Revision	

The ICD revision is encoded in BCD (binary coded decimal). The major code is in the first byte (byte 1) and the minor code is in the second byte (byte 2), e.g. v.3.14 would be encoded as `0x03 0x14`.

The `Silent` flag can be used to disable the blink indication of the PFx Brick status LED when responding to this message. Note that it only disables the blink indication for this message–all other messages will blink the status LED as usual. A value of 1 disables the blink notificaiton, all other values will show the blink indication. This flag is included so that a host can periodically poll the PFx Brick in order to maintain its connection status, without incurring visually distracting status LED blink activity.

4.2 PFX_CMD_GET_STATUS

This command queries the fundamental operational state of the PFx Brick. Normally, the PFx Brick is running its main application firmware. However, the PFx Brick is designed to have its firmware upgraded in the field by the end user with a host PC application. This functionality requires a permanent firmware component called a *bootloader*. The bootloader resides permanently in the PFx Brick and is executed after reset or a power cycle. The bootloader checks to see if valid application firmware has been loaded onto the PFx Brick. If present, it immediately transfers execution to the application firmware. However, if no application firmware is present or corrupted, the bootloader continues to operate the PFx Brick in *Service* mode. This mode has just enough functionality to allow a USB host to load a new application firmware binary image. If successfully loaded, the PFx Brick will restart and then launch the new firmware image.

Additionally, the main application firmware also allows the host to load a new firmware image. In actual fact, the firmware "stages" the new firmware in flash memory, and if successfully loaded, will reboot the PFx Brick. Upon reboot, the bootloader will detect a new "staged" firmware image and attempt to replace the existing firmware with the new one. If successful, the new firmware will execute. If unsuccessful, then at least the PFx Brick will remain in *Service* mode so that another attempt at loading firmware can be made.

One of the goals of the `PFX_CMD_GET_STATUS` command is simply to determine if the PFx Brick is operating normally with its main application firmware or is running the bootloader in Service mode. Based on this determination, the host will know which workflows are permissible. For example, if operating in Service mode, then most USB host commands will simply not work. The only actions that should be exposed to the user are for selecting and loading a new application firmware image.

Lastly, the `PFX_CMD_GET_STATUS` command can be used to determine the specific PFx Brick part number and serial number. This information will be useful for determining the device capabilities, e.g. number of motor channels, storage capacity, etc. as well determining which firmware is compatible with the device. Furthermore, a 24 character product descriptor is included which definitively describes the product identity. Note that the part number and product descriptor are *different* than the USB PID (Product ID). One USB PID may in fact be used to represent a family of PFx Brick procucts. Rather than exhaust the limited availability of USB PID numbers, the Part Number/Product Descriptor pair can be used to detemine the specific PFx Brick type that is connected.

Host command packet:

0	1	2	3	4	5	6	7
0x01	0xA5	0x5A	0x6E	0x40	0x54	0xA4	0xE5

Device response packet:

0	1	2	3 4	5 6	7 8	9 10 11 12
0x81	Status	Error	USB VID	USB PID	Part Number	Serial Number

13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Left justified 24 character product descriptor UTF8 encoded

37 38	39 40
Firmware Version	Firmware Build No.

Status Codes

Status	Code	Description
0x00	PFX_STATUS_NORMAL	if the PFx Brick is running its main application firmware, i.e. normal operation
0x33	PFX_STATUS_NORMAL_PENDING	PFx Brick is running in normal mode with a new application firmware image loaded into staging and pending upgrade
0x55	PFX_STATUS_SERVICE	PFx Brick is running in Service mode with no errors, i.e. a typical state for a new uninitialized PFx Brick
0x53	PFX_STATUS_SERVICE_PENDING	PFx Brick is running in Service mode with a new application firmware image loaded into staging and pending upgrade
0x5B	PFX_STATUS_SERVICE_BUSY	Running in Service mode, busy performing firmware upgrade

Error Codes

Error	Code	Description
0x00	PFX_ERR_NONE	no errors
0x04	PFX_ERR_SPKR_SHORTCIR_FAULT	Short circuit detected on speaker output
0x06	PFX_ERR_TRANSFER_CRC_MISMATCH	Error loading firmware from host into staging memory space
0x08	PFX_ERR_DAC_OVERTEMP_FAULT	Overtemperature condition
0x0B	PFX_ERR_BLE_FAULT	Bluetooth radio module fault
0x80	PFX_ERR_UPGRADE_FAIL	Error copying staged firmware into active operational flash program memory space
0x0A	PFX_ERR_TRAP_BROWNOUT_RST	Reset error due to brownout power condition
0x10	PFX_ERR_TRAP_CONFLICT	Reset error due a trap conflict
0x20	PFX_ERR_TRAP_ILLEGAL_OPCODE	Reset error due to illegal OP code execution
0x40	PFX_ERR_TRAP_CONFIG_MISMATCH	Reset error due configuration mismatch

USB VID/PID

The USB VID (Vendor ID) and PID (Product ID) is part of the standard USB assigned VID/PID pair used to enumerate USB devices.

Serial Number

The serial number is 4 bytes and each PFx Brick will be assigned a unique cryptographically random serial number. The serial number may originate from a unique ID register value embedded in a flash memory device (if available) or it may be assigned by the bootloader after it has been installed.

Firmware Version / Build No.

The firmware version number occupies 2 bytes. The version number is BCD encoded with first byte (byte 37) representing the major version number and the second byte (byte 38) representing the minor version number, e.g. v.3.14 would be encoded as 0x03 0x14. The Build No. complements the version number by indicating a specific build within a series of releases. It is encoded as a verbatim 16-bit value.

Part Number / Part Descriptors

The Part Number is a unique 2-byte value which corresponds to a distinct SKU product. Each product Part Number has a corresponding Product Descriptor. The descriptor is an unambiguous product name encoded as UTF-8 character strings.

Part Number	Product Descriptor	Description
0x1201	PFx Brick alpha	First pre-production prototype PFx Brick with 2x motor channels (using the DRV8839), 8x light channel with discrete pico light connectors, and sound.
0x1202	PFx Brick beta	Second pre-production prototype PFx Brick with 2x motor channels (using the DRV8835), 8x light channels on the standard 10-pin lighting dock connector, and sound.
0x1203	PFx Brick gamma	Third pre-production prototype with 2x motor channels (using the DRV8833), 8x light channels on the standard 10-pin lighting dock connector, and sound.
0x1204	PFx Brick delta IR	Fourth pre-production prototype with 2x motor channels (using the DRV8833), 8x light channels on the standard 10-pin lighting dock connector, and sound.
0x9204	PFx Brick delta	Fourth pre-production prototype with 2x motor channels (using the DRV8833), Bluetooth interface, 8x light channels on the standard 10-pin lighting dock connector, and sound.
0x2204	**PFx Brick IR 4 MB**	Production version of the 4 MB PFx Brick IR with 2x motor channels, 8x light channels, and sound.
0x2208	**PFx Brick IR 8 MB**	8 MB PFx Brick IR
0x2216	**PFx Brick IR 16 MB**	16 MB PFx Brick IR
0xA204	**PFx Brick 4 MB**	Production version of the 4 MB PFx Brick with Bluetooth interface, 2x motor channels, 8x light channels, and sound.
0xA208	**PFx Brick 8 MB**	8 MB PFx Brick
0xA216	**PFx Brick 16 MB**	16 MB PFx Brick
0x1701	PFXLite alpha	Pre-production economy PFx Brick with light f/x only (8x channels with 10-pin dock connector). It has no plastic enclosure, but has stud mounting holes for integration into a model.
0x2702	**PFXLite**	Production economy PFx Brick with light f/x only.
0x1401	PFx Brick Pro alpha	Pre-production PFx Brick with 4x motor channels, 8x light channels, and sound.
0x2404	**PFx Brick Pro 4 MB**	Production 4 MB PFx Brick with 4x motor channels, 8x light channels, and sound.
0x2408	**PFx Brick Pro 8 MB**	8 MB PFx Brick Pro
0x2410	**PFx Brick Pro 16 MB**	16 MB PFx Brick Pro

4.3 PFX_CMD_SET_FACTORY_DEFAULTS

Resets the global configuration, event look-up table and file system with factory default values. This command will overwrite the current configuration of the PFx Brick and cannot be undone.

Host command packet:

0	1	2	3	4	5	6	7
0x02	0xAA	0x55	0xDE	0xAD	0xBE	0xEF	0x02

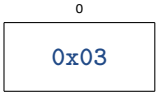
Device response packet:

0
0x82

4.4 PFX_CMD_GET_CONFIG

Retrieves global configuration data from the PFx Brick.

Host command packet:



Device response packet:

Light Ch Brightness

These bytes were formally reserved and are now used to represent individual startup brightness values for each light channel. This includes 8x brightness values for the dedicated light output ports and 2x brightness values for lights attached to the PF Motor channel connectors A and B. Setting individual brightness values is optional. Normally, all channels are set to the master `Default Brightness` value in byte 63. However, if `Default Brightness` is set to zero (0x00), then the individual brightness values for each channel will apply. Having individual default brightness control is useful for situations where relative brightness for each light output is mismatched due to installation, colour, electrical resistance, etc.

Rapid Accel Thr, Rapid Decel Thr, Brake Rate Thr, Brake Speed Thr

These values are used for motor indexed playback of sound effects. These values correspond to either acceleration or speed thresholds which trigger the automated playback of a sound effect. The `Rapid Accel Thr` value specifies the acceleration rate which triggers a sound representing rapid acceleration (e.g. a turbo charger whine sound during heavy acceleration). The `Rapid Decel Thr` value specifies a sound triggered by rapid deceleration (e.g. the sound of dynamic brakes on a locomotive). The `Brake Rate Thr` in combination with te `Brake Speed Thr` value correspond to the acceleration and minimum speed which will trigger playback of a braking sound effect.

Notch Count

The `Notch Count` value specifies how many power "notches" or levels are to be used for simulated engine sound Fx which are indexed by motor speed. This value is only relevant when used with the `SOUND_FX_PLAY_IDX_MOTOR` sound Fx. When this sound Fx is used, up to 8 distinct power levels or notches can be represented by sound files. The selection of a power notch is defined by a desired motor channel's speed. The boundaries between adjacent power notches represent a monotonically changing motor speed. The `Notch 1-2 Bound` represents the motor speed which defines boundary between power notch 1 and 2 and so on. Typically, Notch 1 represents "idle" or minimum motor speed and `Notch Count` represents maximum motor speed. Typically the boundaries between power notches represent evenly spaced intervals of motor speed.

IR Auto Off

The infrared sensor and IR message processing can be configured to automatically turn off and be disabled after a specified interval of time with no activity. This can be a useful feature to either save power or to increase the immunity of the PFx Brick to unintended IR messages.

```
0x00 = Never, IR sensor always enabled
0x01 = Automatic disable after 1 minute of no activity
0x02 = Automatic disable after 5 minutes of no activity
0x03 = Disable immediately after startup (always disabled)
```

BLE Auto Off

The Bluetooth interface can be configured to automatically turn off and be disabled after a specified interval of time with no activity. This can be a useful feature to either save power or reduce radio spectrum congestion.

```
0x00 = Never, BLE interface always enabled
0x01 = Automatic disable after 1 minute of no activity
0x02 = Automatic disable after 5 minutes of no activity
0x03 = Disable immediately after startup (always disabled)
```

BLE Disconnect Motor

If a PFx Brick is being remotely operated by a Bluetooth connected host, there is always the possibility of unintentional disconnection of the radio link due to interference, radio range, or other factors.

When a disconnection occurs, the user has no means of controlling a model until reconnected. In the case of models which are mobile such as trains or cars, this could lead to a "run-away" model situation. In order to avoid this scenario, the PFx Brick can be configured to either continue operating the motors normally or turn off all motors in the event of a BLE disconnection.

```
0x00 = Continue to operate motors normally
0x01 = Turn off all motor channels on a BLE disconnection event
```

BLE Advertisement Power BLE Session Power

The transmitter power of the BLE radio can be adjusted in order to trade-off energy consumption and radio range performance. The BLE radio operates in two basic modes: Advertisement and Connected Session. During Advertisement, the BLE radio will periodically transmit advertisement signals notifying nearby hosts that the PFx Brick is on and available for connection. During a connected session, the BLE radio is used to send messages between the PFx Brick and a connected host for remote control. The transmitter power of both of these modes can be adjusted to trade off energy usage and radio performance.

```
Range between 0x00~0x05 where
0x00 = Maximum transmitter power
0x05 = Minimum transmitter power
```

Audio Bass/Treble

The audio subsystem will have adjustable spectral EQ for bass and treble. The level is specified as a 2's complement signed 8-bit value relative to a nominal value of 0 dB. The adjustable range is therefore -128 to +127 dB; however, in practice it is limited to -20 to +20 dB.

PFX Settings

The PFx Brick has some device specific settings which can be customized by the user. They are encoded as bitfields within the `PFX Settings` byte as follows:

7	6	5 4	3 2	1	0
Reserved	Audio DRC	Lockout/sleep mode	Auto power down mode	Volume beep	Status LED

where

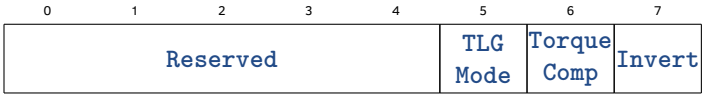
```
Status LED     :  0 = Normally on, wink off with activity
                  1 = Normally off, wink on with activity
Volume Beep    :  0 = No beep sound with change in audio volume
                  1 = Audible beep sound with every change in audio volume
Auto Power
Down Mode      : 00 = No automatic power down
                 01 = Automatic power down/sleep after 30 minutes
                 10 = Automatic power down/sleep after 60 minutes
                 11 = Automatic power down/sleep after 3 hours
Lockout/Sleep
Mode           : 00 = Lockout/sleep disabled
                 01 = Toggle lockout/sleep with 4-double taps on channel 1 only
                 10 = Toggle lockout/sleep with 4-double taps on any channel
                 11 = synonymous with 00 (disabled)

Audio DRC      :  0 = Automatic audio Dynamic Range Control (DRC) off
                  1 = Automatic audio DRC on
```

Motor Configuration

Each motor output on the PFx Brick can be customized by the user for different motor speed and momentum behaviour. These settings apply to each specific motor output connector channel on the PFx Brick. Up to 4x motor channels (A,B,C,D) can be configured; however, the initial version of the PFx Brick has only 2x motor channels fitted (A & B). The settings for channels C & D are placeholders for future 4x channel PFx Bricks.

The motor configuration byte is defined as follows:



where

```
Invert       : 0 = Motor polarity normal
               1 = Motor polarity reversed
               Motors with the same polarity will rotate in the same direction.
Torque Comp  : 0 = High frequency PWM at all speeds (default)
               1 = Low frequency PWM for starting motor with additional torque
                   High freq PWM at all other speeds
TLG Mode     : 0 = Normal high resolution PWM motor control (default)
               1 = Lego IR receiver compatibility mode. Motor driven with low
                   frequency 1 kHz PWM with 7 speed steps in each direction
                   emulating the operation of the Lego IR receiver.
```

vMin, vMid, vMax

These parameters define the shape of the motor speed curve. Normally, motor speed is set directly proportional to user commanded speed (linear). However, this relationship can be modified with alternative speed curves. Examples include parabolically increasing speed curves with more resolution at slower speeds or inverse parabolic curves with rapid initial acceleration. The shape of the curve is a smooth spline-fitted curve between points `vMin`, `vMid`, and `vMax`. `vMin` should be chosen to represent the minimum starting speed of the motor and `vMax` should represent the maximum applied motor speed. Speed values are absolute values between 0 (no speed) up to 255 (maximum speed). This allows the motor to be "clamped" to a maximum speed below the absolute full voltage maximum (255). `vMid` can be chosen to represent the shape of the speed curve. If `vMid` is midway between `vMin` and `vMax`, then the curve will be a standard linear straight line through all three points. If `vMid` is biased toward `vMin`, then the curve will be approximately parabolic with emphasis on low-speed control. Conversely, if `vMid` is biased towards `vMax`, then the speed curve will have an initial rapid increase of speed up to a asymptopic convergence to `vMax`.

Acceleration/Deceleration

The rate at which the user commanded speed and actual motor speed is applied is normally instantaneous. However, momentum or inertia effects can be simulated by setting the acceleration and deceleration factors for increasing and decreasing speed behaviour respectively. For example, a motorized train could have realistic slow acceleration from start and progressive smooth braking to a stop. For no accel/decel effects, these values can be set to 0. Accel/decel factors can be specified from a minimum of 1 up to 255 representing acceleration/deceleration in units of TBD/s.

Default Volume

Configuration for the default audio volume to apply after power up. The valid range is 0x00~0xFF corresponding to minimum and maximum volume respectively.

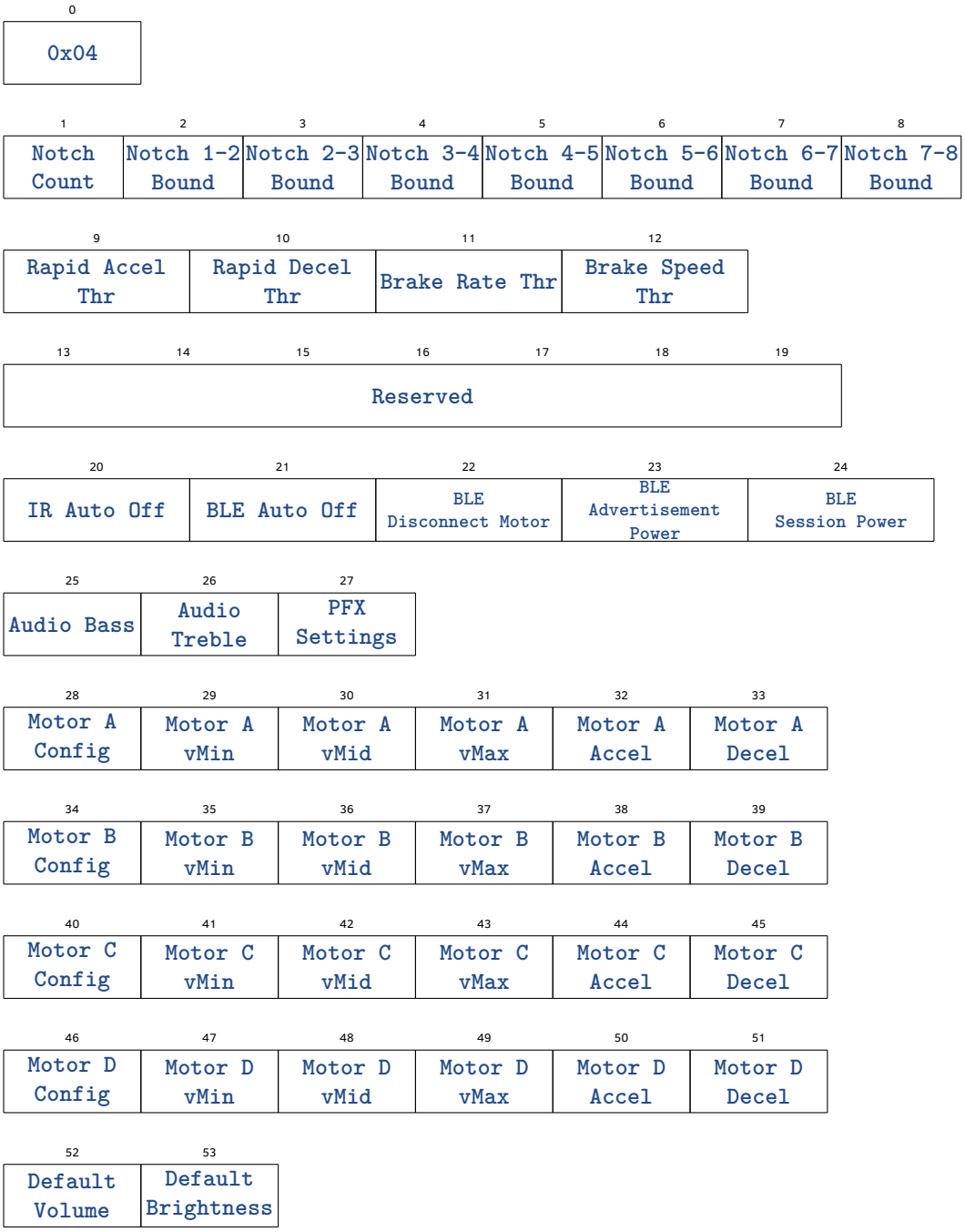
Default Brightness

Configuration for the default global light output brightness to apply after power up. The valid range is 0x00~0xFF corresponding to minimum and maximum brightness respectively.

4.5 PFX_CMD_SET_CONFIG

Overwrites the PFx Brick global configuration data. The PFx Brick will store the new configuration to flash memory.

Host command packet:



54	55	56	57	58	59
Light Ch 1 Brightness	Light Ch 2 Brightness	Light Ch 3 Brightness	Light Ch 4 Brightness	Light Ch 5 Brightness	Light Ch 6 Brightness

60	61	62	63
Light Ch 7 Brightness	Light Ch 8 Brightness	PF Light A Brightness	PF Light B Brightness

Device response packet:

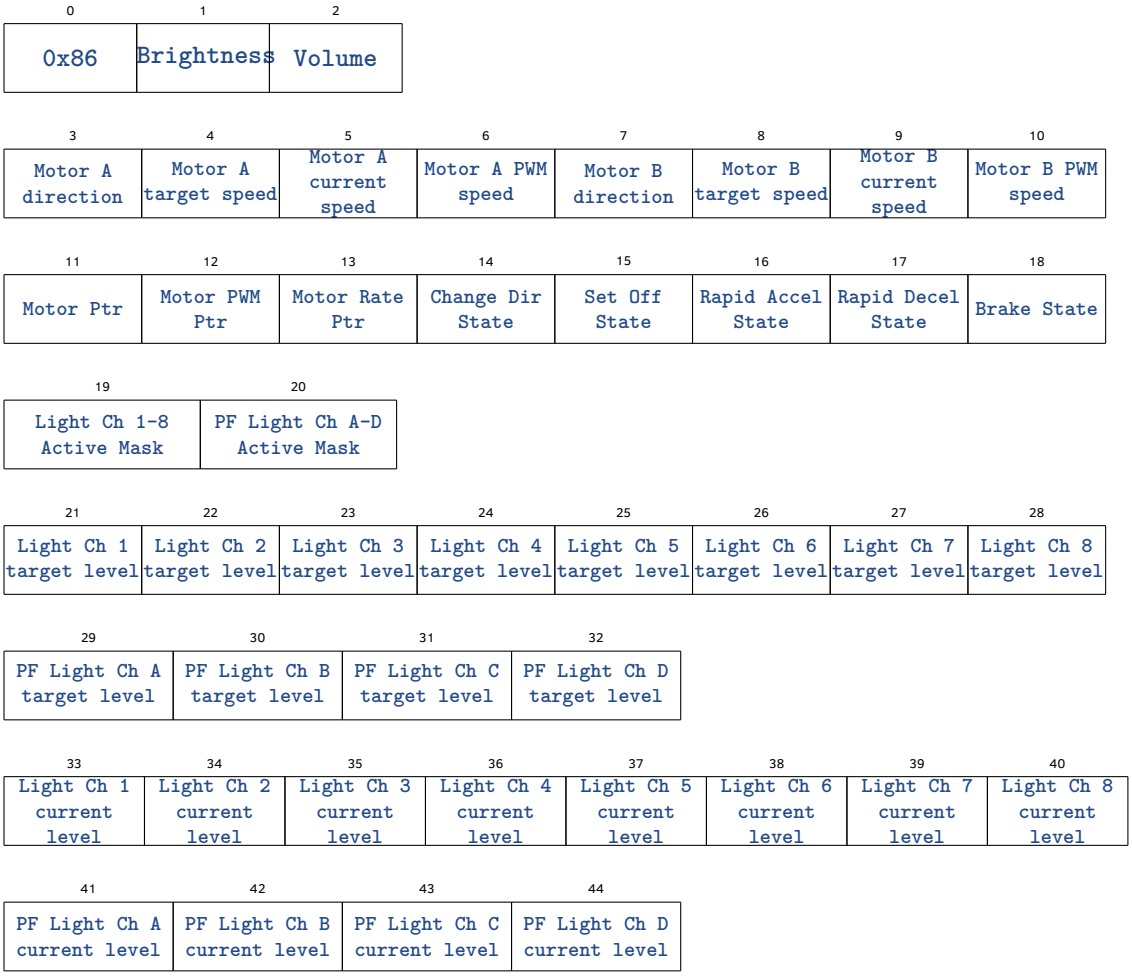
0
0x84

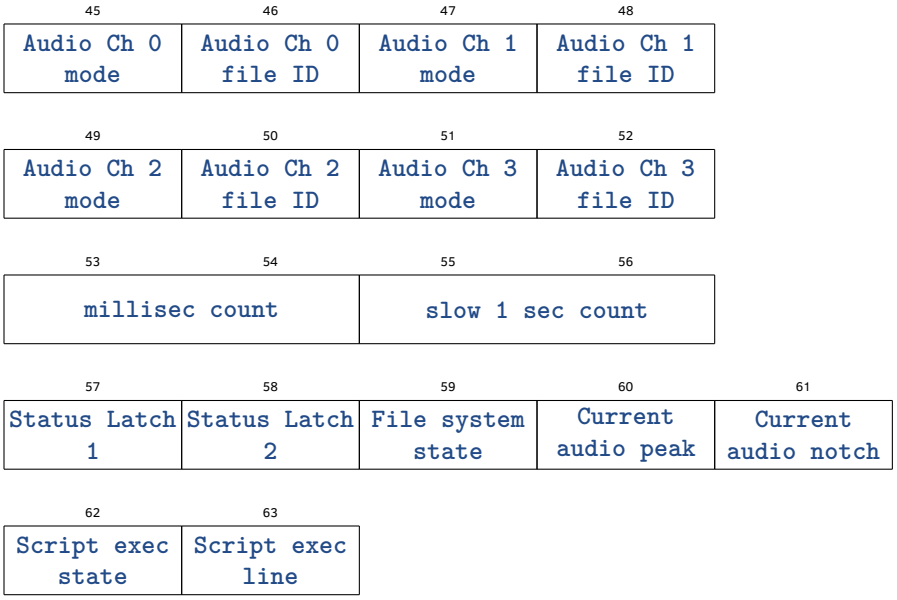
4.6 PFX_CMD_GET_CURRENT_STATE

This message asks the PFx Brick to report its current internal operating state. This includes data such as the current motor target and operating speed, light output states, audio playback status, etc. This information can be useful for test purposes in order to verify that the PFx Brick is correctly responding to event/actions. It is also useful for simple passive monitoring for informational purposes.

Host command packet:

Device response packet:

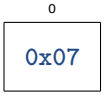



45
Audio Ch 0 mode
46
Audio Ch 0 file ID
47
Audio Ch 1 mode
48
Audio Ch 1 file ID
49
Audio Ch 2 mode
50
Audio Ch 2 file ID
51
Audio Ch 3 mode
52
Audio Ch 3 file ID
53
54
millisec count
55
56
slow 1 sec count
57
Status Latch 1
58
Status Latch 2
59
File system state
60
Current audio peak
61
Current audio notch
62
Script exec state
63
Script exec line

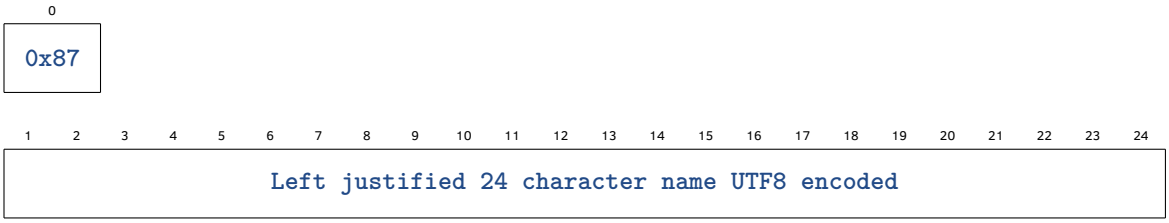
4.7 PFX_CMD_GET_NAME

The device name is user configurable identifier which can be changed at any time. It allows the owner of multiple PFx Bricks to uniquely assign a convenient name for each PFx Brick. The device name is a UTF8 encoded string up to 24 bytes long left justified within the 24 byte block. Unused characters should be padded with zeros (0x00).

Host command packet:



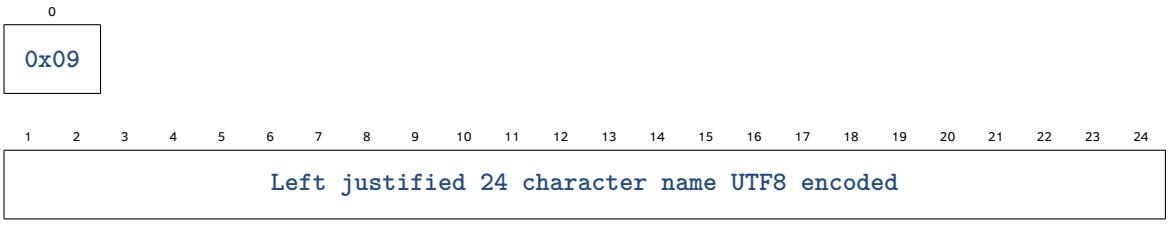
Device response packet:



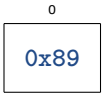
4.8 PFX_CMD_SET_NAME

This message sets the user assigned name of the PFx Brick. The name is 24 bytes long and is UTF8 encoded. Unused characters should be padded with zeros (0x00).

Host command packet:



Device response packet:



4.9 PFX_CMD_GET_EVENT_ACTION

The message allows the host to read the contents of the event LUT for a specific IR remote event and IR channel.

Host command packet:

0	1	2
0x11	Event ID	Channel

or alternatively synonymous with:

0	1	2
0x11	Address [6:2]	Address [1:0]

Device response packet:

0
0x91

1	2
COMMAND	MOTOR_ACTION_ID

3	4
MOTOR_PARAM1	MOTOR_PARAM2

5	6
LIGHT_FX_ID	LIGHT_OUTPUT_MASK

7	8
LIGHT_PF_OUTPUT_MASK	LIGHT_PARAM1

9	10
LIGHT_PARAM2	LIGHT_PARAM3

11	12
LIGHT_PARAM4	LIGHT_PARAM5

13	14
SOUND_FX_ID	SOUND_FILE_ID

15	16
SOUND_PARAM1	SOUND_PARAM2

where the Event ID is defined as:

Event ID	MNEMONIC
0x00	EVT_8879_TWO_BUTTONS
0x01	EVT_8879_LEFT_BUTTON
0x02	EVT_8879_RIGHT_BUTTON
0x03	EVT_8879_LEFT_INC
0x04	EVT_8879_LEFT_DEC
0x05	EVT_8879_RIGHT_INC
0x06	EVT_8879_RIGHT_DEC
0x07	EVT_8885_LEFT_FWD
0x08	EVT_8885_LEFT_REV
0x09	EVT_8885_RIGHT_FWD
0x0A	EVT_8885_RIGHT_REV
0x0B	EVT_8885_LEFT_CTROFF
0x0C	EVT_8885_RIGHT_CTROFF
0x0D	EVT_EV3_BEACON
0x0E	EVT_TEST_EVENT
0x0F	EVT_STARTUP_EVENT
0x10	EVT_STARTUP_EVENT2

Channel is the requested IR channel enumerated as 0,1,2,3 corresponding to the labelled IR channels of 1,2,3,4 respectively. For the EVT_TEST_EVENT the Channel byte is ignored. For the EVT_STARTUP_EVENT the Channel byte specifies one of the four startup events enumerated as 0,1,2,3 corresponding to starup events 1,2,3,4 respectively. Similarly, for EVT_STARTUP_EVENT2 the Channel byte refers to starup events 5,6,7,8.

4.10 PFX_CMD_SET_EVENT_ACTION

The message allows the host to set the contents of the event LUT for a specific IR remote event and IR channel.

Host command packet:

0	1	2
0x12	Event ID	Channel

or alternatively synonymous with:

0	1	2
0x12	Address [6:2]	Address [1:0]

3	4
COMMAND	MOTOR_ACTION_ID

5	6
MOTOR_PARAM1	MOTOR_PARAM2

7	8
LIGHT_FX_ID	LIGHT_OUTPUT_MASK

9	10
LIGHT_PF_OUTPUT_MASK	LIGHT_PARAM1

11	12
LIGHT_PARAM2	LIGHT_PARAM3

13	14
LIGHT_PARAM4	LIGHT_PARAM5

15	16
SOUND_FX_ID	SOUND_FILE_ID

17	18
SOUND_PARAM1	SOUND_PARAM2

Device response packet:

0
0x92

where the Event ID is defined as:

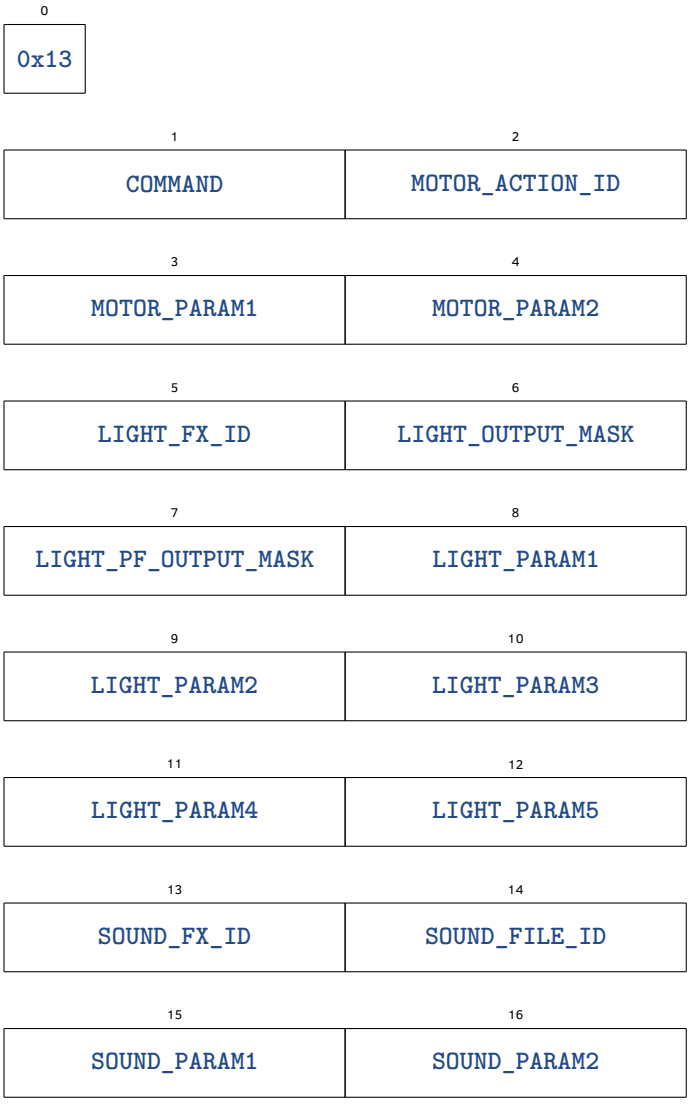
Event ID	MNEMONIC
0x00	EVT_8879_TWO_BUTTONS
0x01	EVT_8879_LEFT_BUTTON
0x02	EVT_8879_RIGHT_BUTTON
0x03	EVT_8879_LEFT_INC
0x04	EVT_8879_LEFT_DEC
0x05	EVT_8879_RIGHT_INC
0x06	EVT_8879_RIGHT_DEC
0x07	EVT_8885_LEFT_FWD
0x08	EVT_8885_LEFT_REV
0x09	EVT_8885_RIGHT_FWD
0x0A	EVT_8885_RIGHT_REV
0x0B	EVT_8885_LEFT_CTROFF
0x0C	EVT_8885_RIGHT_CTROFF
0x0D	EVT_EV3_BEACON
0x0E	EVT_TEST_EVENT
0x0F	EVT_STARTUP_EVENT
0x10	EVT_STARTUP_EVENT2

Channel is the requested IR channel enumerated as 0,1,2,3 corresponding to the labelled IR channels of 1,2,3,4 respectively. For the EVT_TEST_EVENT the Channel byte is ignored. For the EVT_STARTUP_EVENT the Channel byte specifies one of the four startup events enumerated as 0,1,2,3 corresponding to starup events 1,2,3,4 respectively. Similarly, for EVT_STARTUP_EVENT2 the Channel byte refers to starup events 5,6,7,8.

4.11 PFX_CMD_TEST_ACTION

Allows a host to test an event/action. The specified action is performed immediately and is not stored in the event LUT. The format of the action definition is identical to event/actions stored in the event LUT.

Host command packet:

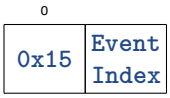


Device response packet:

4.12 PFX_CMD_SEND_EVENT

This message triggers an action from the event/action LUT by specifying an event index into the LUT. The event index corresponds to an equivalent received IR event and can be used to simulate IR events from USB or BLE connected hosts.

Host command packet:



`Event Index` is the address into the event/action LUT. It can also be interpreted as `Event ID` in bits [6:2] and `Channel` in bits[1:0] to form a composite `Event Index` address.

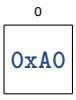
Device response packet:

4.13 PFX_CMD_INC_VOLUME

This message increases the sound volume one increment.

Host command packet:

Device response packet:

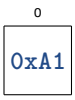


4.14 PFX_CMD_DEC_VOLUME

This message decreases the sound volume one increment.

Host command packet:

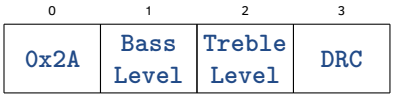
Device response packet:



4.15 PFX_CMD_SET_AUDIO_EQ

This message can be used to set the audio equalization levels for bass and treble as well as setting the state of the automatic Dynamic Range Control (DRC). These values are applied immediately but do not override the default settings stored in the configuration. The values stored in configuration are applied immediately after startup. This message can then be used to set different bass/treble values during operation with a connected USB host.

Host command packet:



Device response packet:

The values for `Bass Level` and `Treble Level` are valid as 2's complement numbers from -20 to 20 inclusive representing the gain/attenuation in dB with a nominal value of 0 dB.

The DRC value is either 0 or 1 reprsenting off or on respectively.

4.16 PFX_CMD_LOAD_FIRMWARE_FILE

This message is the mandatory start message to initiate the transfer of a new firmware image file from the host to the PFx Brick. After this message one or more PFX_CMD_LOAD_FIRMWARE_DATA messages will follow containing the verbatim data content of the firmware image file. Finally, after all of the data has been transferred with multiple PFX_CMD_LOAD_FIRMWARE_DATA messages, a final PFX_CMD_LOAD_FIRMWARE_DONE message is sent to terminate the transfer. After each message, the PFx Brick will respond with an acknowlegement packet to pace the transfer from the host.

The total size of the file in bytes must be specified so that the PFx Brick can pre-allocate the flash memory sectors ahead of the write operations which will follow this message.

This message will not actually replace the running firmware application. Rather, it transfers the new firmware image into a "staging" area. After rebooting the PFx Brick, the bootloader will detect the new firmware image and attempt to replace the existing firmware. The PFX_CMD_REBOOT command can be used to force the reboot process in order to complete the firmware replacement.

PFX Encrypted Firmware Format

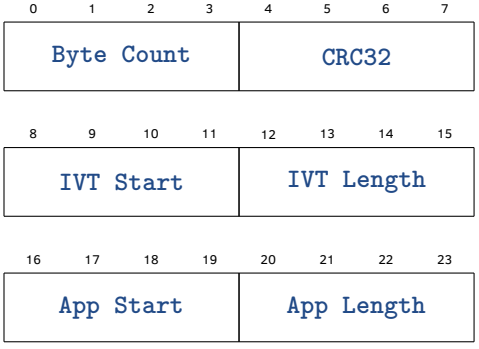
The PFx Brick firmware update process is both secure and robust. This is achieved with 128-bit AES encryption of the firmware payload data and CRC32 verification of the decrypted data. The decryption of the data is performed on the PFx Brick itself so that all data in transit via the USB interface is securely transferred. Furthermore, CRC32 checking is perfomed after transferring the firmware image into its staging area and again after replacing the active firmware image.

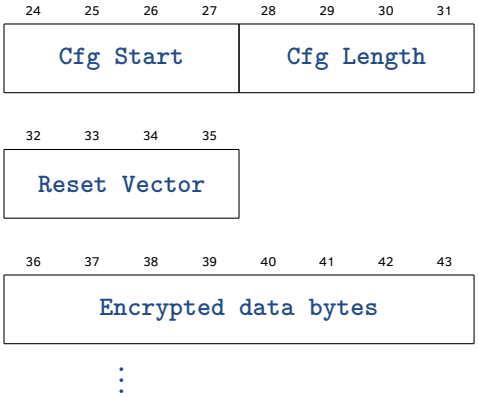
The file format used to transfer firmware image files is a custom format derived from the Intel HEX file format. The PFx Brick firmware is compiled by the Microchip MPLAB X IDE and its linker script generates a standard Intel HEX file describing the firmware application binary. When decoded, this file describes binary data contained in three distinct locations in the PFx Brick microcontroller NVRAM flash memory:

1. IVT Table (Interrupt Vector Table) 0x000 - 0x1FE
2. Application Firmware 0x200-0x1FFFE
3. Configuration Flash Fuses 0xF80000-0xF81000

A CRC32 code is computed over all of the bytes in the IVT and Application Firmware spaces. The Configuration Flash Fuse data is discarded. All of the data bytes in the IVT and Application Firmware spaces are encrypted with AES 128-bit encryption with zero padding if required to acheive an integer multiple of 16 bytes.

The PFX Encrypted Firmware file is then written as follows:





where,

`Byte Count` = total number of data bytes in the IVT and Application firmware spaces

`CRC32` = the CRC32 code computed over the IVT and Application spaces

`IVT Start` = start address of IVT space (word aligned/2-byte boundary)

`IVT Length` = number of 3-byte words in IVT space

`App Start` = start address of Application (word aligned/2-byte boundary)

`App Length` = number of 3-byte words in Application space

`Cfg Start` = start address of Configuration space (word aligned)

`Cfg Length` = number of 3-byte words in Configuration space

`Reset Vector` = start address of application contained at IVT address 0x0000

Host command packet:

0	1	2 3 4 5	6 7 8 9	10 11	12 13 14 15
0x30	File type	Total file size [31:0]	CRC32 [31:0]	IVT size	App size

where,

`File type` = 0 for PFx encrypted Intel HEX file format, 1 for Microchip blob format

`CRC32` is the computed CRC-32 (IEEE 802.3 Ethernet version) over the entire firmware image file. The polynomial implemented is:

x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1

Commonly this is represented as 0xEDB88320 (or 0x04C11DB7 for big endian)

`IVT size` = number of 3-byte words in IVT space `App size` = number of 3-byte words in Application space

Device response packet:

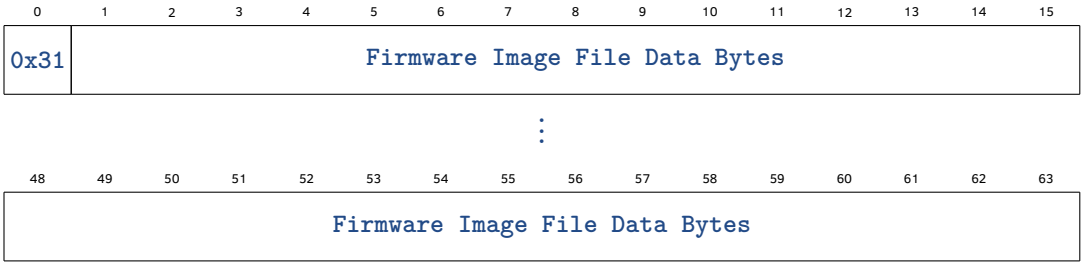
0	1
0xB0	Status

Status	Code	Description
0x00	PFX_ERR_TRANSFER_REQUEST_OK	load firmware file request is ok
0x03	PFX_ERR_TRANSFER_TOO_BIG	file size exceeds free capacity of the firmware staging area

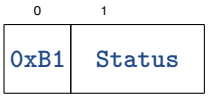
4.17 PFX_CMD_LOAD_FIRMWARE_DATA

One or more of these messages is sent after the PFX_CMD_LOAD_FIRMWARE_FILE message containing the raw byte-for-byte verbatim content of the firmware image file densely packed into every data byte.

Host command packet:



Device response packet:



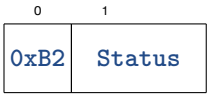
Status	Code	Description
0x00	PFX_ERR_NONE	transfer of firmware payload data ok
0x04	PFX_ERR_TRANSFER_INVALID	data transfer session is invalid (usually due to a missing PFX_CMD_LOAD_FIRMWARE_FILE packet)
0x07	PFX_ERR_TRANSFER_BUSY_WAIT	data transfer of this packet should wait and try again due to an active time-sensitive write or erase operation. The host should reattempt to send the same data packet and check the Status byte.

4.18 PFX_CMD_LOAD_FIRMWARE_DONE

This message is sent after the final PFX_CMD_LOAD_FIRMWARE_DATA message to signal the termination of the firmware file transfer. The host should check the returned error code to ensure that the file transfer was successful.

Host command packet:

Device response packet:



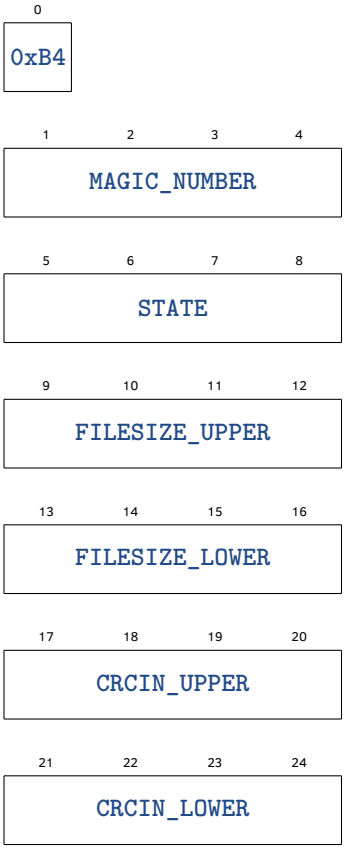
Status	Code	Description
0x00	PFX_ERR_NONE	firmware file transfer completed with no errors
0x04	PFX_ERR_TRANSFER_INVALID	data transfer session is invalid (usually due to a missing PFX_CMD_LOAD_FIRMWARE_FILE packet)
0x06	PFX_ERR_TRANSFER_CRC_MISMATCH	computed CRC32 of received firmware image does not match provided CRC32 code

4.19 PFX_CMD_READ_BOOTCONFIG

This message allows the host to read back the contents of bootloader status and control values stored in the microcontroller NVRAM. These values are used to coordinate the firmware upgrade process between the bootloader and the host as well as storing the operational state of the PFx Brick.

Host command packet:

Device response packet:



4.20 PFX_CMD_REBOOT

Reboots the PFx Brick. This command should only be issued to initiate the upgrade of application firmware after it has been successfully transferred and staged into the PFx Brick.

Note that immediately after issuing this command, the reboot process will terminate the current USB HID communication session. The host application will not be able to communicate with the PFx Brick unless it periodically attempts to re-open a new USB HID session. The host operating system USB stack will continue to re-enumerate the PFx Brick when it restarts and the host application should then be able to re-negotiate a new USB HID session. It will be important for the host application to check the PFx Brick status (i.e. with the PFX_CMD_GET_STATUS command) after re-connection in order to determine whether the PFx Brick is running in Normal mode, Service mode, or if any errors are present in the firmware upgrade process.

Host command packet:

0	1	2	3	4	5	6	7
0x37	0x5A	0xA5	0xD0	0xBE	0xB0	0x04	0x77

Device response packet:

0
0xB7

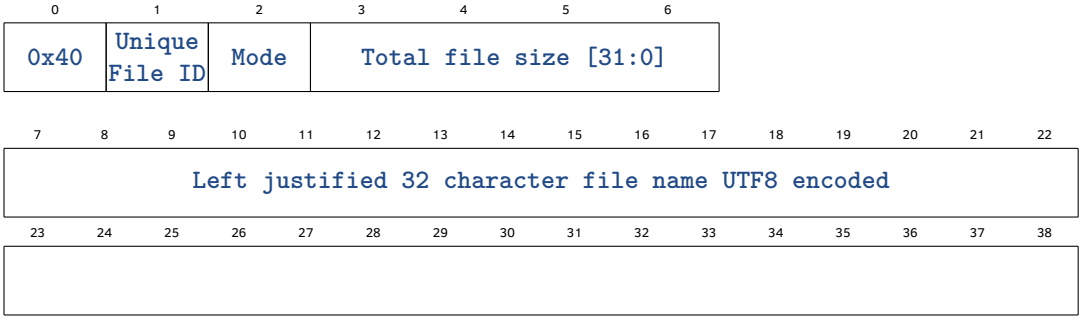
4.21 PFX_CMD_FILE_OPEN

The PFx Brick File System is a simple block-oriented file storage facility which allows files of any content to be transfered to and from the connected host. The primary function of this file system is to store audio files; however, it is general purpose enough to be used for storage of any file type for future applications.

Access to the file system is provided by a set of conventional file I/O methods such as open, close, read, write, etc. Before any file can be accessed, it must be opened. This will ensure that pointers to the file data content for read and write operations are initialized to a known state. Open files must also be closed when the host has completed any read or write tasks. This ensures any buffered data is safely committed back to the file system and the state of file handles and directories remain consistent.

The PFX_CMD_FILE_OPEN command opens a virtual file handle to a file for host file I/O. If the specified file does not exist, then it is created by reserving a directory entry for the file and empty storage sectors are allocated for the file. Unlike other file systems, the creation of a new file requires that the file size be known in advance for preallocation. If the host connects to the PFx Brick via more than one USB HID interface session, each session is granted its own virtual file handle. Futhermore, there is only one file handle per USB HID interface.

Host command packet:



Device response packet:

The Mode parameter is specified as the logical-OR of the following flags:

Mode	Flag	Description
0x01	PFX_FILE_READ	open file with read access
0x02	PFX_FILE_WRITE	open file with write access
0x04	PFX_FILE_CREATE	create a new file with ID and size

If a new file is created with PFX_FILE_CREATE mode flag, then the specified file ID must be unique and the total file size must be specified in bytes. Optionally, a 32 character UTF-8 filename can be specified with the file create request. This name appears in the file directory. If the name is not specified, the request will still succeed and the file can be renamed at any other time after it is created. If the file ID is already in use, then the file open request will not succeed. File open requests

on existing files (without the create flag) only need to specify the file ID and do not need to specify file name or size.

If the file specified by ID is valid, then a virual file handle will be retained on the PFx associated with the USB interface channel that made the request. This file handle can then be used to perform subsequent read and write file operations.

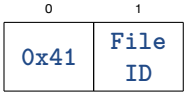
The file open request will return a status code which indicates either success or error according to the table below. Note that these error codes are shared among all of the file system access commands and returned in the `Status` byte. These error codes are also repeated in the Error Code section at the end of this document.

Status	Code	Description
0x00	PFX_ERR_NONE	file system operation ok
0xF0	PFX_ERR_FILE_SYSTEM_ERR	overall file system error
0xF1	PFX_ERR_FILE_INVALID	file request was invalid or file is invalid
0xF2	PFX_ERR_FILE_OUT_OF_RANGE	file access request is outside of file size
0xF3	PFX_ERR_FILE_READ_ONLY	file creation or write access denied
0xF4	PFX_ERR_FILE_TOO_BIG	requested file creation is too big
0xF5	PFX_ERR_FILE_NOT_FOUND	requested file ID is not found
0xF6	PFX_ERR_FILE_NOT_UNIQUE	requested file creation ID is already used
0xF7	PFX_ERR_FILE_LOCKED_BUSY	file system is locked or busy
0xF8	PFX_ERR_FILE_SYSTEM_FULL	file system full
0xF9	PFX_ERR_FILE_SYSTEM_TIMEOUT	file access operation time out
0xFA	PFX_ERR_FILE_INVALID_ADDRESS	file system request resulted in an invalid memory address
0xFB	PFX_ERR_FILE_NEXT_SECTOR	file system FAT points to an invalid sector
0xFC	PFX_ERR_FILE_ACCESS_DENIED	file system operation denied or prohibited
0xFF	PFX_ERR_FILE_EOF	file access has reached the end of the file

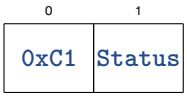
4.22 PFX_CMD_FILE_CLOSE

The PFX_CMD_FILE_CLOSE command closes the virtual file handle to a file which was opened with the PFX_CMD_FILE_OPEN command. It is important to close a file especially after any write operations. This is to ensure that any buffered or cached data is committed to the file system so that no written data is lost.

Host command packet:



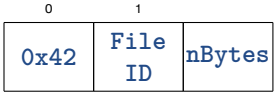
Device response packet:



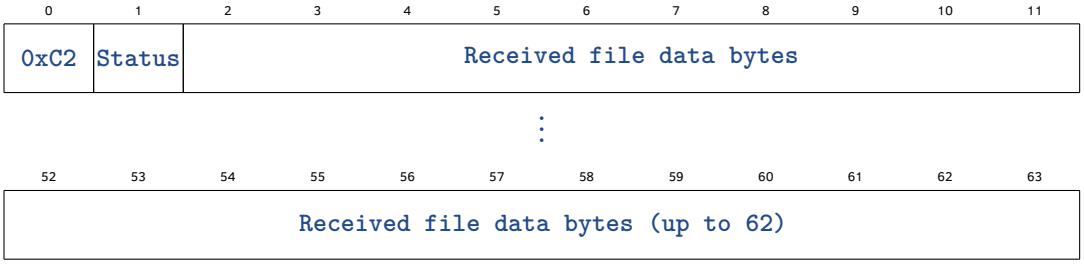
4.23 PFX_CMD_FILE_READ

The PFX_CMD_FILE_READ command is used to read file data sequentially from the current file read pointer location. Each read file operation advances the file pointer by how many file bytes have been retrieved. This ensures consecutive read operations maintain continuity along the file data stream.

Host command packet:



Device response packet:



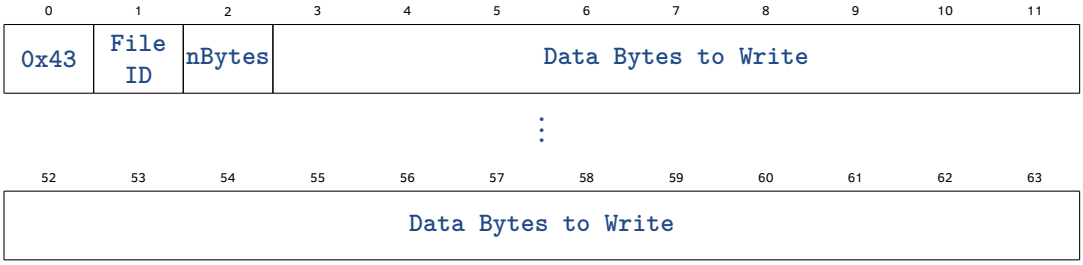
The nBytes field specifies up to how many data bytes should be read (valid range is 1-62).

The returned Status byte is either an error code or the number of bytes (1-62) contained in this packet.

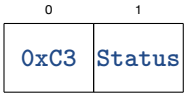
4.24 PFX_CMD_FILE_WRITE

The PFX_CMD_FILE_WRITE command is used to write file data sequentially from the current file write pointer location. Each write file operation advances the file pointer by how many file bytes have been written. This ensures consecutive write operations maintain continuity along the file data stream.

Host command packet:



Device response packet:



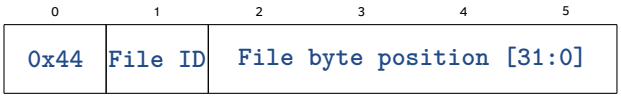
The nBytes field specifies up to how many data bytes should be read (valid range is 1-62).

The returned Status byte error code indicates if write operaton was successful.

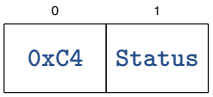
4.25 PFX_CMD_FILE_SEEK

The PFX_CMD_FILE_SEEK command is used to reposition the file access pointer to any location within the file. The position is specified as an absolute value in bytes relative to the start of the file.

Host command packet:



Device response packet:



4.26 PFX_CMD_FILE_DIR

The PFX_CMD_FILE_DIR command is used to interact with the file system directory. The file directory contains a list of files currently stored on the file system along with several attributes and data fields. This command can be used request different types of directory information such as the number of files, free space, individual file directory entries, etc. It can also be used to modify the directory entry of a stored file.

Host command packet:

0	1	2
0x45	Request	File ID, index, optional parameters, ...

The Request byte can be specified as follows:

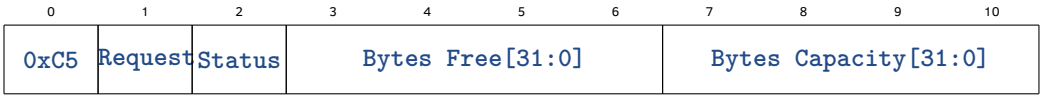
Status	Code	Description
0x00	PFX_DIR_REQ_GET_FILE_COUNT	Get number of files
0x01	PFX_DIR_REQ_GET_FREE_SPACE	Get free space and total capacity
0x02	PFX_DIR_REQ_GET_DIR_ENTRY_IDX	Get directory entry at index
0x03	PFX_DIR_REQ_GET_DIR_ENTRY_ID	Get directory entry of File ID
0x04	PFX_DIR_REQ_ADD_AUDIO_FILE_ID	Add audio meta data to directory for File ID
0x05	PFX_DIR_REQ_RENAME_FILE_ID	Rename File ID
0x06	PFX_DIR_REQ_SET_ATTR_ID	Set attributes for File ID
0x07	PFX_DIR_REQ_SET_USER_DATA1_ID	Set UserData1 attributes for File ID
0x08	PFX_DIR_REQ_SET_USER_DATA2_ID	Set UserData2 attributes for File ID
0x09	PFX_DIR_REQ_COMPUTE_CRC32_ID	Compute CRC32 for File ID
0x0A	PFX_DIR_REQ_SET_ATTR_MASKED_ID	Set attributes with mask for File ID
0x0B	PFX_DIR_REQ_GET_NAMED_FILE_ID	Get File ID for file name
0x0C	PFX_DIR_REQ_GET_SMALL_DIR_ID	Get compact file info of File ID
0x0D	PFX_DIR_REQ_GET_SMALL_DIR_IDX	Get compact file info at index
0x0E	PFX_DIR_REQ_CHANGE_FILE_ID	Change File ID

Device response packets

Request 0x00 - Get Number of Files

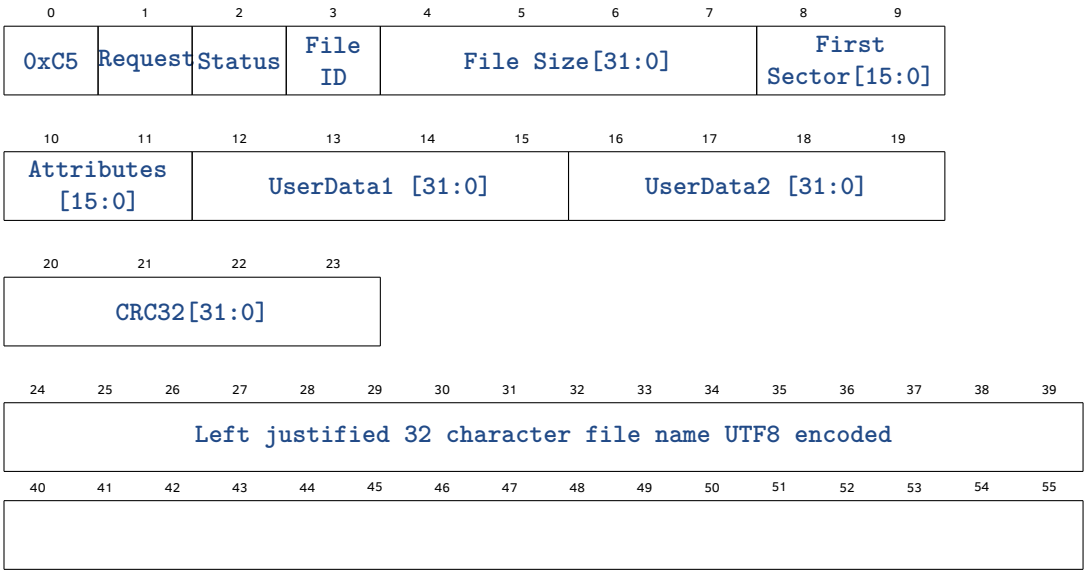
0	1	2	3
0xC5	Request	Status	File Count[15:0]

Request 0x01 - Get Free Space / Capacity



Request 0x02 - Get Directory Entry at Index

Request 0x03 - Get Directory Entry of File ID



Request 0x04 - Add Audio Meta Data to Directory with ID

This command will trigger the file system to read the specified file and extract meta data associated with an audio WAV file. This meta data is then written to the directory in the `Attributes`, `UserData1`, and `UserData2` fields.

Request 0x05 - Rename File with ID

Changes the 32 character filename of the specified file. The filename data bytes should be contained in bytes 3 to 34 of the host command packet.

Request 0x06 - Set Attributes with ID

Changes the `Attributes` field of the file directory entry. The `Attributes[15:0]` data bytes should be contained in bytes 3 and 4 of the host command packet.

Request 0x0A - Set Attributes with ID, masked

Changes the `Attributes` field of the file directory entry. The `Attributes[15:0]` data bytes should be contained in bytes 3 and 4 of the host command packet and a bit mask should be contained in bytes 5 and 6. The only bits that are changed in the `Attributes` field are the bits specified with the bit mask. This allows non-destructive modification of attributes by only specifying the bits that require changing. For example a command to modify the file type of file ID `0x77` to WAV would be as follows: `0x45 0x0A 0x77 0x00 0x00 0xFF 0x00`, i.e. only `User Attributes[15:8]` is set to `0x00` because of the bit mask `0xFF00`.

Request 0x07 - Set UserData1 with ID

Request 0x08 - Set UserData2 with ID

Changes the UserData1/2 fields of the file directory entry. The UserData1/2[31:0] data bytes should be contained in bytes 3 to 6 of the host command packet.

Request 0x09 - Compute CRC32 with ID

Computes the CRC32 hash code of the specified file and stores it into the file directory. Normally, the CRC32 code is automatically computed when a file that is being written is closed. This command can be used force the recalculation of the CRC32 code. Note that the computation of the CRC32 code is performed as a background process and may take several seconds to complete for large files. The CRC32 code is set to zero before a new computation is performed. This can be used to monitor the progress of the CRC32 computation since it will revert to a non-zero value when it is completed.

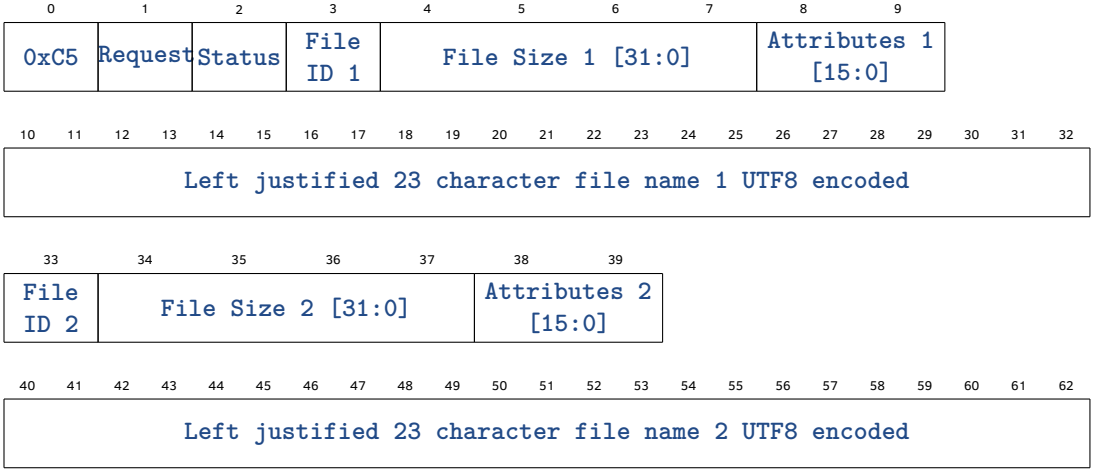
Request 0x0B - Get File ID for Filename

Attempts to find the file ID of a specified filename. The filename data bytes should be contained in bytes 3 to 34 of the host command packet and byte 2 should contain the length of filename. If the filename is found, then it is returned in the Status field, otherwise an error code indicating PFX_ERR_FILE_NOT_FOUND or PFX_ERR_FILE_NOT_UNIQUE may be returned.

Request 0x0C - Get Compact File Info with ID

Request 0x0D - Get Compact File Info with Index

This message returns two consecutive file directory entries in a compact form within one message. This directory request type is available for the benefit of Bluetooth mobile hosts needing to enumerate sound files quickly by reducing the number of BLE transactions and bandwidth. The message returns only essential file information such as size, attributes, and a truncated version of the filename. The message also packs two entries starting at the index of requested file and if it exists, the next consecutive file directory entry.



Request 0x0E - Change File ID

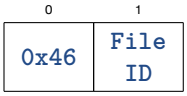
This message changes the file ID of an existing file. Byte 2 is file ID to change and byte 3 contains the new file ID.

The `Status` byte contains the result code of the directory operation request which should nominally be 0x00 indicating success.

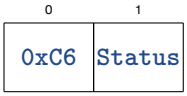
4.27 PFX_CMD_FILE_REMOVE

The PFX_CMD_FILE_REMOVE command deletes a file from the file system. The file is specified by its unique File ID.

Host command packet:



Device response packet:



4.28 PFX_CMD_FILE_FORMAT_FS

The PFX_CMD_FILE_FORMAT_FS command erases and re-initializes the entire file system. After this command is performed, the PFx Brick will automatically start to pre-erase the file storage space on the flash memory. During this process, the host can continue to access the file system; however, response times will be reduced due to the arbitration that must take place to interleave access to the flash memory. The process of pre-erasing memory usually takes less than one minute and after it is completed, full response time will be restored.

Host command packet:

0	1	2	3	4
0x47	0xEA	0x5E	0x88	Mode

Device response packet:

0	1
0xC7	Status

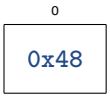
The Mode parameter can be used to specify one of two formatting modes:

```
0 = Fast Format:  erases only occupied sectors
1 = Complete:  erases all sectors
```

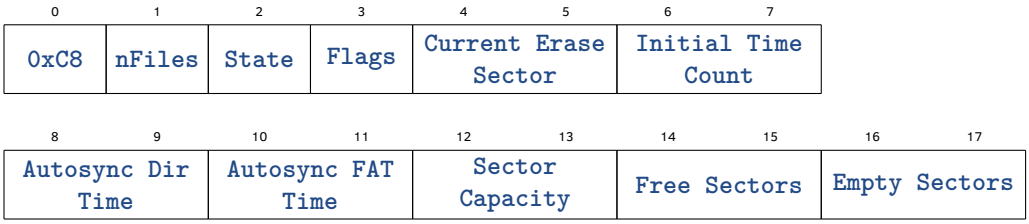
4.29 PFX_CMD_FILE_GET_FS_STATE

The PFX_CMD_FILE_GET_FS_STATE command reports low-level operational status information of the file system. This data is mainly used for test and debug purposes; however, it could be used for useful status updates.

Host command packet:



Device response packet:



The nFiles byte reports the number of files contained in the file system.

The State byte reports the state of the finite state machine which operates the file system.

The Flags byte reports operational state flags of the file system.

The Current Erase Sector field reports the current sector of the garbage collection process. This value will change continuously representing the on-going scanning of FAT looking for freed sectors to erase.

The Initial Time Count field reports the initial timer value of time out counters.

The Autosync Dir Time and Autosync FAT Time fields report the timer values of the autosync hold-off before any autosync processes commit file system changes to flash memory.

Sector Capacity reports the total available storage capacity in 4096 byte sectors of the file system. The total byte capacity can be computed by multiplying this value by 4096.

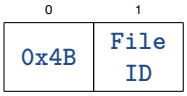
Free Sectors reports the sum of free and empty sectors. Sectors are 4096 byte storage blocks of the file system. The free byte capacity can be computed by multiplying this value by 4096. When a file is removed or if the file system is formatted, occupied sectors are de-allocated from the file system and marked as free. These free sectors can be made available for storage after the file system recovers the sectors by erasing them in an automated garbage collection process. After free sectors are erased, they become empty sectors available for re-allocation for new files.

Empty Sectors reports the remaining available empty sectors. Empty sectors can be allocated for the creation of new files. The available byte capacity can be computed by multiplying this value by 4096.

4.30 PFX_CMD_RUN_SCRIPT

The PFX_CMD_RUN_SCRIPT command starts execution of a script file stored in the file system. The file is specified by its unique File ID.

Host command packet:



Device response packet:

The Status byte contains the result code of this command and should nominally be 0x00 indicating success.

If the File ID byte is set to 0xFF, then the current running script will be stopped.

4.31 PFX_CMD_STATUS_LED

This message allows the host to either poll or set the state of the status LED.

Host command packet:

0	1	2
0x70	get=0 set=1	on=1 off=0

To get the state of the LED, byte 1 is 0. To set the state of the LED, byte 1 is non-zero, and byte 2 turns the LED off if 0, and on otherwise.

Device response packet:

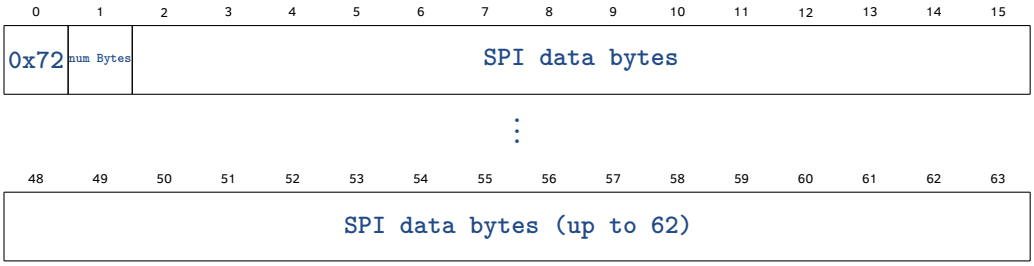
0	1
0xF0	LED State

LED state = 0 if LED is off, non-zero if LED is on.

4.32 PFX_CMD_WRITE_SPI

This message allows the host to perform a write command over the SPI bus connected to the flash memory. This permits very low level access to the flash memory device for test and debug purposes.

Host command packet:



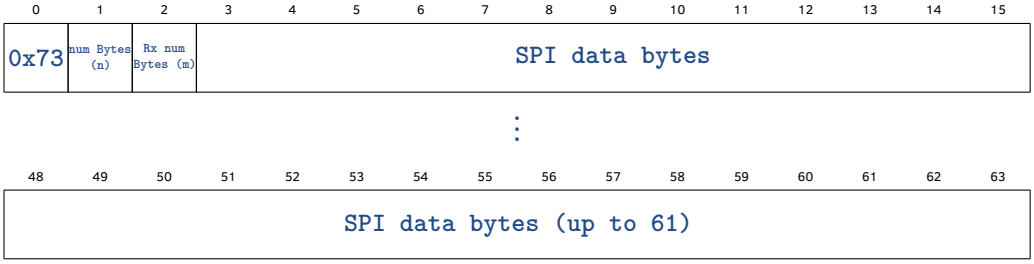
`numBytes(n)` specifies how many payload SPI bytes are contained in this packet (<=62) each byte in the desired SPI transfer follows up to the specified numBytes.

Device response packet:

4.33 PFX_CMD_READ_SPI

This message allows the host to perform a write command over the SPI bus and read back a corresponding SPI response. This permits very low level access to the flash memory device for test and debug purposes.

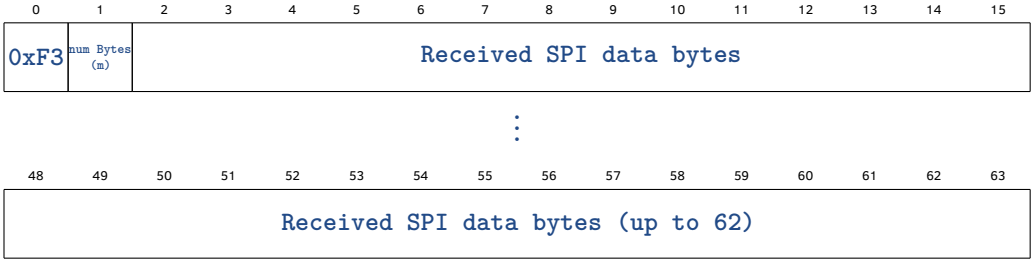
Host command packet:



`numBytes(n)` specifies how many payload SPI bytes are contained in this packet (<=61)

`numBytes(m)` specifies how many response SPI bytes are expected in return (<=62) each byte in the desired SPI transfer follows up to the specified numBytes.

Device response packet:

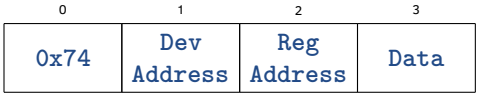


`numBytes(m)` specifies how many response payload SPI bytes are contained in this packet (<=62)

4.34 PFX_CMD_WRITE_I2C

This message allows the host to perform a write command over the I2C bus. This permits very low level access to connected I2C devices such as the audio DSP/DAC for test and debug purposes.

Host command packet:

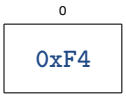


`Dev Address` is the I2C 7-bit device address of the audio DSP/DAC device. Normally this is 0x30 for the Texas Instruments TLV320DAC3120 fitted to the PFx Brick.

`Reg Address` is the address of the register within the I2C device that is desired to be accessed.

`Data` is the value to write to the specified I2C register.

Device response packet:



4.35 PFX_CMD_READ_I2C

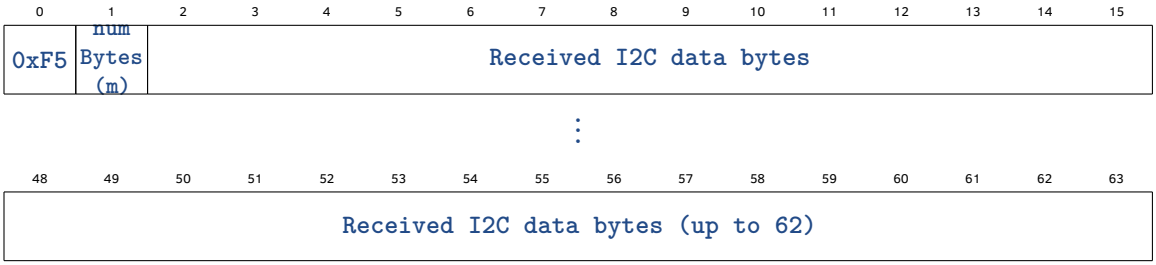
This message allows the host to read a device register over the I2C bus. This permits very low level access to connected I2C devices such as the audio DSP/DAC for test and debug purposes.

Host command packet:

`Dev Address` is the I2C 7-bit device address of the audio DSP/DAC device. Normally this is 0x30 for the Texas Instruments TLV320DAC3120 fitted to the PFx Brick.

`Reg Address` is the address of the register within the I2C device that is desired to be accessed.

Device response packet:

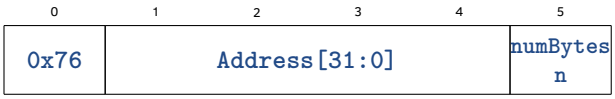


`numBytes(m)` specifies how many response payload I2C bytes are contained in this packet (<=62)

4.36 PFX_CMD_READ_FLASH

This message allows the host to read back the contents of the flash memory device starting at specified address up to 63 additional byte locations.

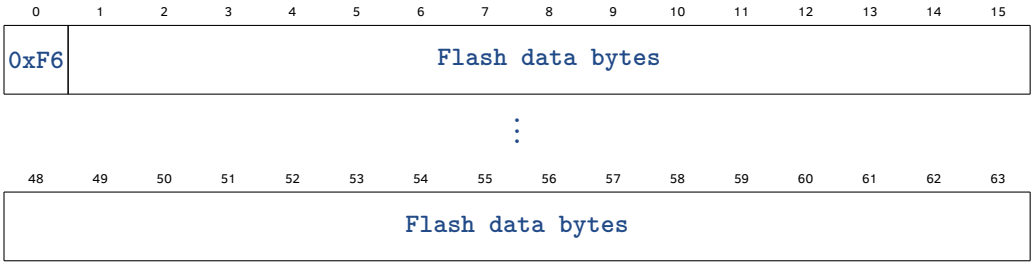
Host command packet:



`Address[31:0]` is a 32-bit byte aligned address

`numBytes(n)` specifies how many bytes to read starting at address (1<=n<64)

Device response packet:



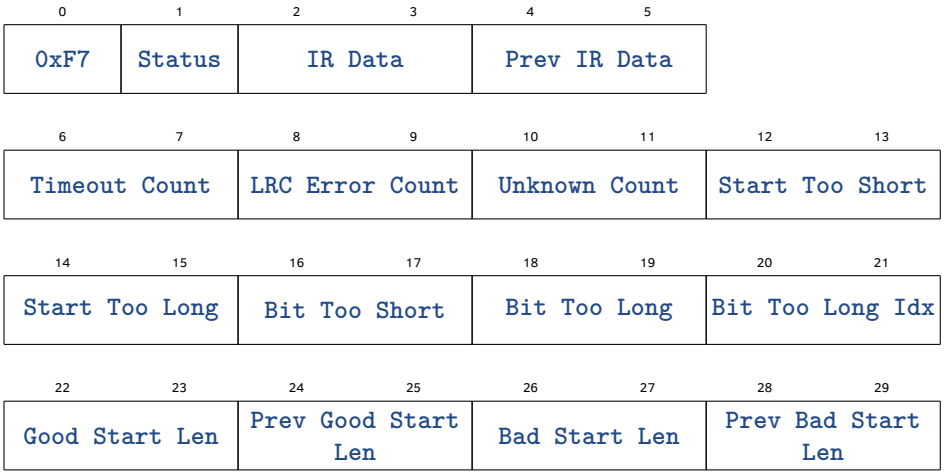
byte 1 is data as read from `Address[31:0]` byte 2 is data as read from `Address[31:0]+1` and so on

4.37 PFX_CMD_GET_IRRX_STATUS

This command retreives detailed low level data from the IR receiver protocol processor. This message may or may not be supported for a particular PFx Brick due to the overhead required to capture the data. The return message from the PFx Brick will indicate if there is valid data available.

Host command packet:

Device response packet:



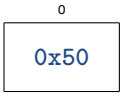
`Status` is 1 if IR protocol data is available in bytes 6-29 contained in this message. If `Status` is 0, then bytes 6-29 do not contain valid data since it is unsupported by the version of PFx Brick queried.

The `IR Data` and `Prev IR Data` fields are always valid independent of the value of `Status`.

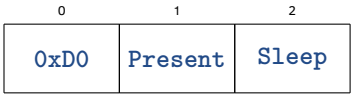
4.38 PFX_CMD_GET_BT_STATUS

This command gets the operational status of the Bluetooth interface module.

Host command packet:



Device response packet:



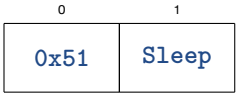
`Present` = 0 if no Bluetooth interface is installed, 1 = Bluetooth interface available

`Sleep` = 0 if Bluetooth module is active, 1 = Bluetooth module is in power saving sleep mode

4.39 PFX_CMD_SET_BT_POWER

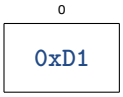
This command sets the power mode of the Bluetooth interface module.

Host command packet:



`Sleep` = 1 puts the Bluetooth interface module into low power sleep mode and disables the Bluetooth module. Setting `Sleep` to 0 wakes up the Bluetooth module for normal operation.

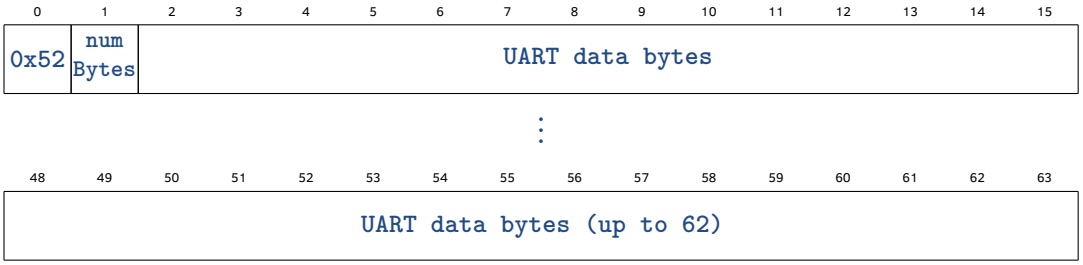
Device response packet:



4.40 PFX_CMD_SEND_BT_UART

This command sends an ASCII message to the Bluetooth interface module UART.

Host command packet:



numBytes specifies how many payload bytes are contained in this packet (<=62) each byte in the desired UART message follows up to the specified numBytes.

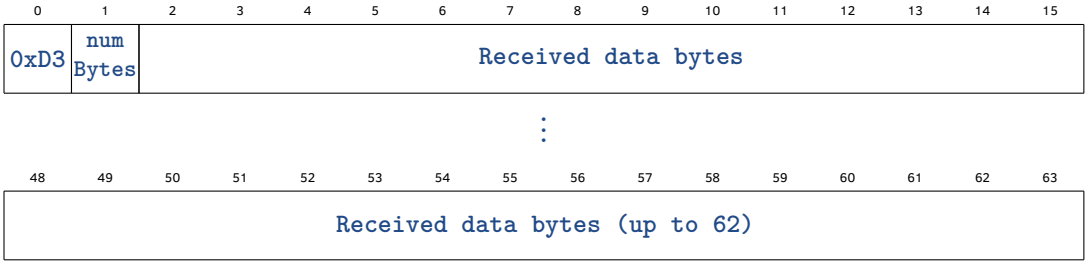
Device response packet:

4.41 PFX_CMD_RECEIVE_BT_UART

This message reads back the contents of the receive buffer from the Bluetooth module UART.

Host command packet:

Device response packet:



numBytes specifies how many bytes are contained in this packet (<=62)

4.42 PFX_CMD_SET_NOTIFCATIONS

This message configures the notification service in the PFx Brick.

Host command packet:

0	1
0x60	Flags

A detailed description of the `Flags` field can be found in section 7 of this document.

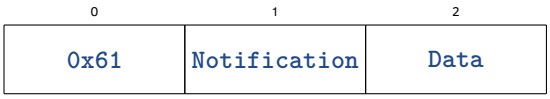
Device response packet:

0
0xE0

4.43 PFX_MSG_NOTIFICATION

These messages are sent asynchronously from the PFx Brick after notifications have been configured by a connected host using the PFX_CMD_SET_NOTIFCATIONS command. Each notification message contains information about one notification event. Therefore, if the host subscribes to two or more notifcations, then multiple notification messages can be expected from the PFx Brick, one or more for each event. Unlike the command to set notifications which represent the logical-OR of multiple notifications, the notification messages themselves are sent individually, one for each desired notification.

The format of the notification message from the PFx Brick is as follows:



The Notification field represents the notification type. The Data field optionally contains extra qualifying data if applicable.

Notification	MNEMONIC	Data
0x01	PFX_NOTIFICATION_AUDIO_PLAY_DONE	File ID of audio file
0x02	PFX_NOTIFICATION_AUDIO_PLAY	File ID of audio file
0x04	PFX_NOTIFICATION_MOTORA_CURR_SPD	Current motor speed
0x08	PFX_NOTIFICATION_MOTORA_STOP	n/a
0x10	PFX_NOTIFICATION_MOTORB_CURR_SPD	Current motor speed
0x20	PFX_NOTIFICATION_MOTORB_STOP	n/a

5 Scripting Actions

As of ICD version 3.37 (and PFx Brick firmware versions 1.40+), the ability to execute complex actions and behaviours defined in script files was added. Script files are simple, human readable text files stored in the PFx Brick file system. These files conform to a simple script language syntax described later in this document. The scripting capability can be summarized as follows:

1. Scripts are ASCII text files stored in the PFx Brick file system.
2. Scripts execute one at a time. Executing another script will terminate the current script and start the new one.
3. Scripts can be executed either by using an Event/Action (with the `COMMAND` byte) or with the ICD message `PFX_CMD_RUN_SCRIPT`.
4. Script execution is sequential line-by-line from the start of the file to the end. At the end, the script will either stop or repeat if a repeat command is the last line.
5. Script lines with bad syntax are ignored and script execution will continue to the next line.

5.1 Loading Scripts

Script files can be loaded on to the PFx Brick using the PFx App or by using other 3rd party software to copy files from a host PC to the PFx Brick. A script file will have a name and file ID on the PFx Brick file system. The unique file ID must be known in order to execute a script (see the file system section for more information).

Creating script files or making changes to a script must be made on a host PC using any standard text editor (e.g. Windows Notepad, macOS Text Editor, etc.) To modify a script file, the old one must be removed from the PFx Brick file system and then replaced with a new copy (with the same file ID).

5.2 Executing Scripts

Script files can be executed in one of three ways:

1. Using an Event/Action
2. Using the `PFX_CMD_RUN_SCRIPT` ICD message via USB or BLE.
3. Automatically after power on or reset if the filename is `startup.pfx`

5.2.1 Event/Action Script Execution

The Event/Action data structure `COMMAND` byte (0) can be set to `COMMAND_RUN_SCRIPT` (0x09) and the script file ID can be specified in the `SOUND_FILE_ID` byte (13). When a IR remote action is configured this way, it will trigger the execution of the specified script file. Therefore, a simple event from a remote control can trigger a very complex sequence of actions defined by the script.

5.2.2 `startup.pfx` Automatic Script Execution

If a script file is copied to the PFx Brick file system with the special reserved filename of `startup.pfx`, then it will execute automatically after power on or reset. This can be useful for assigning special configuration or complex startup action sequences common. Examples include train locomotive profiles which assign audio files for specific roles, configure motor parameters and configure actions for special lighting effects and sound playback.

5.2.3 ICD Message

The PFX_CMD_RUN_SCRIPT ICD message can be sent to the PFx Brick via a USB or BLE connected host to trigger the execution of a script file. A unique file ID must be specified in the message to indicate which script file to execute.

5.3 Script Syntax

The PFx Brick script language syntax is a simple human readable free form text file format. Script files can contain comments and arbitrary amounts of whitespace in addition to the recognized script keywords. Script file execution is sequential and proceeds line by line from the start of the file to the end. This implies that all logical script commands must be terminated with a either a linefeed (0x0A) and/or carriage return character (0x0D).

5.3.1 Comments

Comment lines start with either a # character (similar to python) or // characters (similar to C++). Comments should not be used in line with a command.

```
# Valid comment
// Another valid comment
light 1 on  # not a valid comment location
```

5.3.2 Keywords

The script syntax uses case sensitive keyword commands and specifiers. There are several primary keywords which act as commands and many secondary keywords used for specifying subcommands, parameters values and options.

The primary keyword commands are as follows:

```
event
ir parameters
light channels commands
motor channels commands
repeat
run
set
sound commands
stop
wait parameters
```

The secondary command and parameter keywords are as follows:

```
acc, all, bass, beep, ble, bright, button, ch, changedir, config,
connect, decel, disconnect, down, fade, file, flash, fx, gated,
invert, joy, left, loop, long, off, on, play, rate, right, servo,
shutdown, speed, startup, thr, treble, up, vol
```

5.3.3 Numeric Values

Many commands and options require specified numeric quantities. The script syntax supports both integer and decimal values. The following are examples of valid numeric quantities:

```
0 127 -55 0.010 35.75 -90.5
```

Additionally, integer values may be specified in hexadecimal (base16) prefixed with the characters `0x`.

```
0x0 0xABCD 0x32
```

For commands which support a list of values, a list is specified as a group of comma separated numbers enclosed in matching square brackets:

```
[0, 1, 2, 3]
```

5.3.4 Strings

Some commands also support the use of strings–typically for specifying items such as filenames. Strings are UTF-8 formatted and enclosed within double quotations marks `"`.

```
"This is a string"
```

5.3.5 User Variables

There are 6 fixed name variables which act like storage registers named $A, $B, $C, $D, $E, and $F. The preceding dollar sign ($) is required and the variable names must be uppercase.

Variables are assigned values with the set keyword, the variable name followed by an equals sign (=) and the value to assign. Variables can be assigned any numeric value or string.

```
set $A = 50
motor a speed $A
set $B = "Beep1.wav"
sound play $B
```

5.3.6 Repeat Loops

A block of script code can be repeated a specified number of times using the repeat keyword. The code block must be encapsulated in curly braces { }. The opening brace { must appear on the same line as the repeat keyword and the closing brace } must appear on a line by itself to mark the end of the code block.

Repeated code blocks can be nested up to four levels.

```
repeat 5 {
  light 1 fx 1
  repeat 3 {
    light 2 fx 1
    wait 2.0
  }
}
```

5.3.7 Event Action Configuration

The event command allows a script to assign actions associated with an event. The events correspond to the same events in the event LUT. When a script changes an event LUT entry it is stored in the PFx Brick non-volatile memory and is changed in the same way as using the PFX_CMD_SET_EVENT_ACTION host command. The event command syntax is as follows:

```
event type {
  ...
}
```

The desired actions are enclosed between the braces. The opening brace must be on the same line as the event command and the closing brace must be on a line by itself. Actions specified between the braces are not performed during script execution. They are stored and only executed when the desired event occurs. Multiple actions can be associated with an event; however, only one of each type can stored, i.e. only one light, sound, and motor action.

5.4 Command Reference

Light Commands

`light` ***channels commands***

channels can be specified as a single channel number 1-8, a list of channels enclosed with [] parenthesis, or the keyword `all`

commands are a combination of the following keywords and values:
`on` - turn on light channel(s)
`off` - turn off light channel(s)
`fade` ***<time>*** - fade time (0 to 10.0 seconds).
`flash` ***<ontime>*** [***offtime***] - periodic flashing light (0.05 to 60.0 seconds)
`bright` ***<value>*** - set brightness (0 to 255)
`fx` ***<id>*** [***parameters***] - performs light action ***<id>*** as `LIGHT_FX_ID` with specified parameters
if ***channels*** = `all` then ***<id>*** is a combo id

Sound Commands

`sound` ***command***

command is one of the following keywords:
`play` ***fileID*** - start playback of ***fileID***
`stop` ***fileID*** - stop playback of ***fileID***
`play` ***fileID*** `repeat` - continuous playback of ***fileID***
`play` ***fileID*** `loop` ***<value>*** - plays ***fileID*** for ***value*** times
`vol` ***<value>*** - set volume (0 to 255)
`bass` ***<value>*** - set bass (-20 to 20)
`beep` - short beep sound
`treble` ***<value>*** - set treble (-20 to 20)
`fx` ***<id>*** ***fileID*** [***parameters***] - performs sound action ***<id>*** as `SOUND_FX_ID` with specified parameters

fileID can be specified either as a numeric file ID or string containing the filename.

Motor Commands

`motor channels command`

channels can be specified as a single channel number 1 or 2 (or as `a` and `b`), a list of channels enclosed with `[]` parenthesis, or the keyword `all`

command is one of the following keywords:
`stop` - stop motor channel(s)
`speed <value>` - motor speed (-255 to 255), +speed is forward, -speed is reverse direction
`servo <value>` - servo motor angle (-90 to 90)
`acc <value>` - optional acceleration/deceleration (0 to 15)
`fx <id> [parameters]` - performs motor action `<id>` as `MOTOR_FX_ID` with specified parameters

---

IR Commands

`ir on` - activates the IR sensor
`ir off` - disables the IR sensor

Set Command

`set var = value`
assigns any of the 6 variables `$A`, `$B`, `$C`, `$D`, `$E`, or `$F` to a numeric or string value.

`set config parameter = value`
sets a configuration setting stored in the PFx Brick non volatile memory.
Resulting behaviour is similar to using the `PFX_CMD_SET_CONFIG` ICD command message.

parameter can be specified as:
`set config bass = value` - audio bass -20 to +20
`set config treble = value` - audio treble -20 to +20
`set config vol = value` - default audio volume 0 to 255
`set config bright chan = value` - light channel brightness *chan*: 1 to 8 *value*: 0 to 255
`set config nc = value` - number of notches for motor indexed playback, 1 to 8
`set config nb notch = value` - notch boundary speed *notch*: 1 to 8 *value*: 0 to 255

`set config motor chan accel = value` - motor acceleration *chan*: a or b *value*: 0 to 15
`set config motor chan decel = value` - motor decelerration *chan*: a or b *value*: 0 to 15
`set config motor chan invert = value` - invert motor polarity *chan*: a or b *value*: 0 to 1
`set config motor chan v0 = value` - speed curve min *chan*: a or b *value*: 0 to 255
`set config motor chan v1 = value` - speed curve mid *chan*: a or b *value*: 0 to 255
`set config motor chan v2 = value` - speed curve max *chan*: a or b *value*: 0 to 255

`set config thr accel = value` - Threshold for rapid accel sound 0 to 255
`set config thr decel = value` - Threshold for rapid decel sound 0 to 255
`set config thr rate = value` - Threshold accel for braking sound
`set config thr speed = value` - Threshold speed for braking sound 0 to 255

`set file type = fileID`
assigns files used for motor indexed playback and gated playback actions.

fileID can be specified with either a numeric file ID or string.
type can be specified as:
`set file speed notch = fileID` - Sound loop at speed notch *notch*
`set file accel notch = fileID` - Sound loop for accel between notches *notch*
`set file decel notch = fileID` - Sound loop for decel between notches *notch*
`set file gated notch = fileID` - Gated sound loops (up to 4x for ea. notch 1 to 4)
notch can be specified as 11 to 14, 21 to 24, 31 to 34, or 41 to 44

`set file startup = fileID` - Sound played at startup
`set file shutdown = fileID` - Sound played at shutdown
`set file changedir = fileID` - Sound played due to direction change
`set file thr accel = fileID` - Sound played due to rapid accel
`set file thr decel = fileID` - Sound played due to rapid decel
`set file brake on = fileID` - Sound played with brake application
`set file brake off = fileID` - Sound played at set-off from stop

Event Command

`event type {`
`...`
`}` - stores action(s) within encapsulated code block to event

type can be specified as:
`event address {` - indexed by address in the even LUT 0x00 to 0x7C
`event startup chan {` - startup event 1 to 8
`event ir parameters {` - IR event defined the same way as `ir` command
`event button {` - button press event
`event button long {` - long button press event
`event button down {` - button state transitioned to down
`event button up {` - button state transitioned to up
`event ble connect {` - Bluetooth host session started
`event ble disconnect {` - Bluetooth host session ended
`event usb connect {` - USB host session started
`event usb disconnect {` - USB host session ended

Execution Control

Delay execution and wait for event to resume:
`wait` *<time>* - pause (0.05 to unlimited sec)
`wait sound` *fileID* - pause execution until sound file *fileID* has stopped playing
`wait button` - wait for a push button event
`wait ir` *parameters* - pause execution until IR event has been received
where *parameters* can be any combination of:
`joy` - joystick remote, `speed` - speed remote, `up,down,left,right,button` - remote actions
`ch` *<value>* - IR channel

Repeat Loops:
`repeat` - repeat execution of current script
`repeat` *<value>*`{`
`...`
`}` - repeat execution of encapsulated code block *value* times

Redirect execution to same or different script:
`run` *fileID* - execute script with *fileID* (numeric or string)
`stop` - stops the script at the current line

5.5 Examples

```
# Traffic light sequence
#
# Ch 1: Red, Ch 2: Yellow, Ch 3: Green
# Ch 4: Don't Walk, Ch 5: Walk
# reset all light channels
light all off
# Red phase
light [1,4] on
light [2,3,5] off fade 0.2
wait 8.0
# Green phase
light [1,4] off fade 0.2
light [3,5] on
wait 8.0
# Pedestrian crossing warning
light 5 off fade 0.1
light 4 flash 0.4 fade 0.1
wait 5
# Yellow
light 3 off fade 0.2
light [2,4] on
wait 4
# Start the sequence again
repeat
```

```
# Motorized musical procession
#
# Vehicle with motor, lights and music; Triggered by IR remote

# Start with everything off
light all off
sound stop all
motor all off

# Wait for joystick remote ch 1 right up
wait ir joy ch 1 right up

# Play sound and move
motor a speed 30
light all on
sound play "MySong.wav"

# Wait until song is finished, stop and repeat
wait sound "MySong.wav"
motor a stop
repeat
```

```
#
# Using loops to make lighting effects
#

light all off
# store delay interval in A
set $A = 0.2
# store repeat count in B
set $B = 4

# This light sequence will be repeated 5 times:
# light 1 will toggle followed by 4 toggles of light 2
repeat 5 {
  light 1 on
  wait $A
  light 1 off
  wait $A
  repeat $B {
    light 2 on
    wait $A
    light 2 off
    wait $A
  }
}
```

```
#
# Configuring actions associated with Bluetooth status
#

# When a remote host connects via Bluetooth
# - play a sound
# - disable IR sensor
event ble connect {
  sound play "Welcome.wav"
  ir off
}

# When a remote host ends Bluetooth session
# - play a sound
# - enable IR sensor
event ble disconnect {
  sound play "Goodbye.wav"
  ir on
}
```

6 Event/Action Data Structures

The fundamental behaviour of the PFx Brick is to perform actions in response to received IR and/or Bluetooth interface events. Actions are encoded in a data structure called the event look up table (LUT). The actions performed are indexed by a corresponding event trigger into the event LUT, i.e. the event LUT is “addressed” by message events. This section will describe event LUT and the many associated fields and parameters.

6.1 Event Encoding

The events sent by IR remotes and/or Bluetooth interface cue corresponding actions stored in the event look up table. These actions include controlling motors, light f/x and sound. Some event actions may depend on the current state of other items, e.g. the change of direction on a motor channel may depend on its current speed.

The event LUT address format used internally by the PFx Brick is as follows:

LEGO® Power Funcitons IR Remotes

Address	Event ID	MNEMONIC
0x00-0x03	0x00	EVT_8879_TWO_BUTTONS
0x04-0x07	0x01	EVT_8879_LEFT_BUTTON
0x08-0x0B	0x02	EVT_8879_RIGHT_BUTTON
0x0C-0x0F	0x03	EVT_8879_LEFT_INC
0x10-0x13	0x04	EVT_8879_LEFT_DEC
0x14-0x17	0x05	EVT_8879_RIGHT_INC
0x18-0x1B	0x06	EVT_8879_RIGHT_DEC
0x1C-0x1F	0x07	EVT_8885_LEFT_FWD
0x20-0x23	0x08	EVT_8885_LEFT_REV
0x24-0x27	0x09	EVT_8885_RIGHT_FWD
0x28-0x2B	0x0A	EVT_8885_RIGHT_REV
0x2C-0x2F	0x0B	EVT_8885_LEFT_CTROFF
0x30-0x33	0x0C	EVT_8885_RIGHT_CTROFF
0x34-0x37	0x0D	EVT_EV3_BEACON

Following the Power Functions IR remote events, there are special event LUT entries reserved for other purposes as follows:

Address	Event ID	MNEMONIC	Description
0x38	0x0E	EVT_TEST_EVENT	Used for testing actions sent by a USB connected host
0x3C	0x0F	EVT_STARTUP_EVENT1	Used for storing start-up actions performed after power on
0x3D	0x0F	EVT_STARTUP_EVENT2	
0x3E	0x0F	EVT_STARTUP_EVENT3	
0x3F	0x0F	EVT_STARTUP_EVENT4	
0x40	0x10	EVT_STARTUP_EVENT5	
0x41	0x10	EVT_STARTUP_EVENT6	
0x42	0x10	EVT_STARTUP_EVENT7	
0x43	0x10	EVT_STARTUP_EVENT8	
0x44	0x11	EVT_BUTTON_PRESS	Used for defining push button actions from a touchLAB
0x45	0x11	EVT_BUTTON_LONGPRESS	
0x46	0x11	EVT_BUTTON_DOWN	
0x47	0x11	EVT_BUTTON_UP	

LEGO® RC Train IR Remote

The RC Train remote was black with 4 yellow buttons. The buttons are labelled Up, Down, Horn, and Stop. A channel selector switch allows you to select channels labelled 1, 2, 3, 1+2+3. These channels correspond to 0, 1, 2, 3 respectively within the event LUT.

Address	Event ID	MNEMONIC
0x50-0x53	0x14	EVT_RCTRAIN_UP
0x54-0x57	0x15	EVT_RCTRAIN_DOWN
0x58-0x5B	0x16	EVT_RCTRAIN_STOP
0x5C-0x5F	0x17	EVT_RCTRAIN_HORN

Sparkfun COM-11759 Mini IR Remote

Address	Event ID	Ch	MNEMONIC
0x60	0x18	0	EVT_SPARKFUN_POWER
0x61	0x18	1	EVT_SPARKFUN_A
0x62	0x18	2	EVT_SPARKFUN_B
0x63	0x18	3	EVT_SPARKFUN_C
0x64	0x19	0	EVT_SPARKFUN_UP
0x65	0x19	1	EVT_SPARKFUN_DOWN
0x66	0x19	2	EVT_SPARKFUN_LEFT
0x67	0x19	3	EVT_SPARKFUN_RIGHT

Adafruit 389 Mini IR Remote

Address	Event ID	Ch	MNEMONIC
0x68	0x1A	0	EVT_ADAFRUIT_VOLDOWN
0x69	0x1A	1	EVT_ADAFRUIT_PLAY
0x6A	0x1A	2	EVT_ADAFRUIT_VOLUP
0x6B	0x1A	3	EVT_ADAFRUIT_SETUP
0x6C	0x1B	0	EVT_ADAFRUIT_STOP
0x6D	0x1B	1	EVT_ADAFRUIT_UP
0x6E	0x1B	2	EVT_ADAFRUIT_DOWN
0x6F	0x1B	3	EVT_ADAFRUIT_LEFT
0x70	0x1C	0	EVT_ADAFRUIT_RIGHT
0x71	0x1C	1	EVT_ADAFRUIT_ENTER
0x72	0x1C	2	EVT_ADAFRUIT_REPEAT
0x73	0x1C	3	EVT_ADAFRUIT_0
0x74	0x1D	0	EVT_ADAFRUIT_1
0x75	0x1D	1	EVT_ADAFRUIT_2
0x76	0x1D	2	EVT_ADAFRUIT_3
0x77	0x1D	3	EVT_ADAFRUIT_4
0x78	0x1E	0	EVT_ADAFRUIT_5
0x79	0x1E	1	EVT_ADAFRUIT_6
0x7A	0x1E	2	EVT_ADAFRUIT_7
0x7B	0x1E	3	EVT_ADAFRUIT_8
0x7C	0x1F	0	EVT_ADAFRUIT_9

6.2 Action Encoding

The event LUT stores encoded actions in a multi-byte data structure. The actions performed by the PFx Brick are grouped into the following categories:

1. Motor Actions
2. Single Light F/X Output Actions
3. Combo Light F/X Output Actions
4. Sound F/X Actions

Note that these actions can be combined to respond to a single event, e.g. play a sound with a lighting effect, actuate multiple lights as a group, etc.

The encoded action data structure is composed of 16 bytes as follows:

	7	6	5	4	3	2	1	0
0	COMMAND							
1	MOTOR_ACTION_ID				MOTOR_MASK			
2	MOTOR_PARAM1							
3	MOTOR_PARAM2							
4	COMBO	LIGHT_FX_ID						
5	LIGHT_OUTPUT_MASK							
6	LIGHT_PF_OUTPUT_MASK							
7	LIGHT_PARAM1							
8	LIGHT_PARAM2							
9	LIGHT_PARAM3							
10	LIGHT_PARAM4							
11	LIGHT_PARAM5							
12	SOUND_FX_ID							
13	SOUND_FILE_ID							
14	SOUND_PARAM1							
15	SOUND_PARAM2							

When an event is triggered (e.g. from an IR remote, or a PFX_CMD_TEST_ACTION message is received via USB or BLE), the PFx Brick performs the action specified by the associated action data structure. The action is processed sequentially starting from the first byte COMMAND.

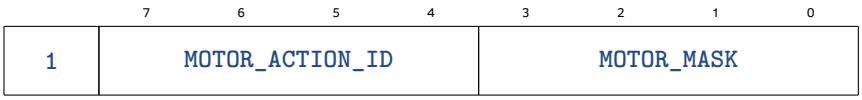
1. If COMMAND in byte 0 is non-zero, the specified command is executed and **rest of the action data structure is ignored**. One execption is when the COMMAND byte is specified as COMMAND_RUN_SCRIPT; in this case the PFx Brick will execute the script file specified in the SOUND_FILE_ID byte (13).
2. If MOTOR_MASK in byte 1 is non-zero, the motor action specified by MOTOR_ACTION_ID is performed using the parameters MOTOR_PARAM1 and MOTOR_PARAM2.
3. If LIGHT_FX_ID in byte 4 is non-zero, the light effect action is performed on the light outputs specified by LIGHT_OUTPUT_MASK and LIGHT_PF_OUTPUT_MASK using the parameters in LIGHT_PARAM1-5.
4. If SOUND_FX_ID in byte 12 is non-zero, the sound effect action is performed with the sound file specified by SOUND_FILE_ID using parameters SOUND_PARAM1 and SOUND_PARAM2.

6.2.1 COMMAND

The COMMAND byte is used to perform special actions not related to the core actions related to motors, lights and sound. The supported commands are listed as follows:

ID	MNEMONIC	Description
0x00	COMMAND_NONE	No action
0x01	COMMAND_ALL_OFF	Shut off all motor channels, light output ports, and stop all audio playback
0x02	COMMAND_IR_LOCKOUT_ON	Activate IR receiver lockout, ignores IR receiver packets
0x03	COMMAND_IR_LOCKOUT_OFF	Deactivate IR receiver lockout, resumes processing of IR receiver
0x04	COMMAND_IR_LOCKOUT_TOGGLE	Toggle the state of the IR lockout mode
0x05	COMMAND_ALL_MOTORS_OFF	Turn off all motor channels
0x06	COMMAND_ALL_LIGHTS_OFF	Turn off all lighting channels
0x07	COMMAND_ALL_AUDIO_OFF	Stop all audio playback
0x08	COMMAND_RESTART	Stop all current actions, and restart with all STARTUP actions
0x09	COMMAND_RUN_SCRIPT	Execute a script file specified by the file ID in the SOUND_FILE_ID byte (13)

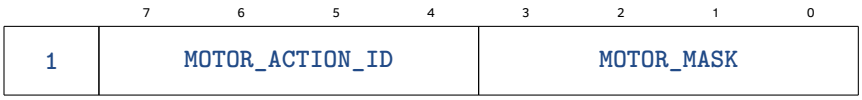
6.2.2 MOTOR_ACTION_ID



The MOTOR_ACTION_ID is a 4-bit encoded value which occupies bits [7:4] and specifies the type of action to apply to the motor channel(s) specified in the MOTOR_MASK bits. The MOTOR_ACTION_ID bits are defined as follows:

ID	MNEMONIC	Description
0x00	MOTOR_ESTOP	Motor braked to stop immediately (emergency stop)
0x01	MOTOR_STOP	Motor commanded to stop using the configured deceleration rate.
0x02	MOTOR_INC_SPEED	Increase motor speed one step (up to vMax)
0x03	MOTOR_DEC_SPEED	Decrease motor speed one step (clamped to zero)
0x04	MOTOR_INC_SPEED_BIDIR	Increase motor speed one step; passing zero changes direction
0x05	MOTOR_DEC_SPEED_BIDIR	Decrease motor speed one step; passing zero changes direction
0x06	MOTOR_CHANGE_DIR	Change motor direction (motor must be stopped first)
0x07	MOTOR_SET_SPD	Sets the motor speed to specific value
0x08	MOTOR_SET_SPD_TIMED	Sets the motor speed to run for a fixed time
0x09	MOTOR_OSCILLATE	Oscillate motor speed on and off
0x0A	MOTOR_OSCILLATE_BIDIR	Oscillate motor speed forward/reverse
0x0B	MOTOR_OSCILLATE_BIDIR_WAIT	Oscillate motor speed forward/reverse with a wait interval in between
0x0C	MOTOR_RANDOM	Set random motor speed periodically within set speed
0x0D	MOTOR_RANDOM_BIDIR	Set random motor speed and direction periodically within set speed
0x0E	MOTOR_SOUND_MODULATED	Set motor speed within set speed modulated by sound intensity
0x0F	MOTOR_SET_SERVO	Set LEGO Power Functions servo motor position

6.2.3 MOTOR_MASK



Motor actions can be applied to any combination of motor outputs simultaneously, e.g. two or more motors controlled to the same speed. The MOTOR_MASK<3:0> has 4 bits corresponding to motor outputs D,C,B,A respectively. The initial PFx Brick design has only 2 motor outputs (A & B); however, provision for future expanded versions of the PFx Brick with 4 motor outputs is being accomodated. A bit value of 1 in each position indicates that the corresponding motor output will be controlled, e.g. MOTOR_MASK<3:0>=0xA means that motor outputs D and B will be operated together.

6.2.4 MOTOR_PARAMx

	7	6	5	4	3	2	1	0
2	MOTOR_PARAM1							
3	MOTOR_PARAM2							

The MOTOR_PARAM1 and MOTOR_PARAM2 bytes encode parameters which are associated with some of the MOTOR_ACTION_ID items. The definition of MOTOR_PARAM1 and MOTOR_PARAM2 is shown in the table below:

ID	MNEMONIC	MOTOR_PARAM1	MOTOR_PARAM2
0x00	MOTOR_ESTOP		
0x01	MOTOR_STOP		
0x02	MOTOR_INC_SPEED	MOTOR_STEP	
0x03	MOTOR_DEC_SPEED	MOTOR_STEP	
0x04	MOTOR_INC_SPEED_BIDIR	MOTOR_STEP	
0x05	MOTOR_DEC_SPEED_BIDIR	MOTOR_STEP	
0x06	MOTOR_CHANGE_DIR		
0x07	MOTOR_SET_SPD	MOTOR_SPEED	
0x08	MOTOR_SET_SPD_TIMED	MOTOR_SPEED	DURATION
0x09	MOTOR_OSCILLATE	MOTOR_SPEED	MOTOR_PERIOD
0x0A	MOTOR_OSCILLATE_BIDIR	MOTOR_SPEED	MOTOR_PERIOD
0x0B	MOTOR_OSCILLATE_BIDIR_WAIT	MOTOR_SPEED	MOTOR_PERIOD
0x0C	MOTOR_RANDOM	MOTOR_SPEED	MOTOR_PERIOD
0x0D	MOTOR_RANDOM_BIDIR	MOTOR_SPEED	MOTOR_PERIOD
0x0E	MOTOR_SOUND_MODULATED	MOTOR_SPEED	
0x0F	MOTOR_SET_SERVO	MOTOR_POS	

6.2.4.1 MOTOR_SPEED The MOTOR_SPEED parameter specifies absolute motor speed to be directly applied without intermediate incremental steps. This parameter is defined as follows:

MOTOR_SPEED	Value	
0x0	stopped	
0x1	10%	
0x2	25%	% of maximum
0x3	33%	speed in the
0x4	50%	forward direction
0x5	67%	
0x6	75%	
0x7	100%	
0x8	stopped	
0x9	10%	
0xA	25%	% of maximum
0xB	33%	speed in the
0xC	50%	reverse direction
0xD	67%	
0xE	75%	
0xF	100%	

The MOTOR_SPEED parameter can also be used to specify a higher resolution set speed. This is achieved by setting the MOTOR_SPEED[7] bit to '1' and using the MOTOR_SPEED[6] bit as a direction flag. MOTOR_SPEED[5:0] bits specify absolute speed in either direction. Therefore MOTOR_SPEED can be defined as follows for high resolution speed settings:

7	6	5 4 3 2 1 0
1	Dir 0=fwd, 1=rev	Speed

This allows for a range of speed settings as follows:

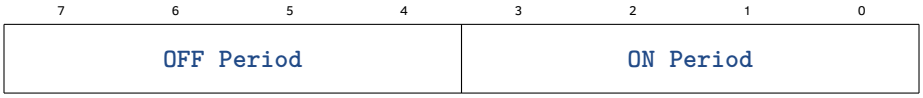
MOTOR_SPEED		Speed
0xBF	1 0 1 1 1 1 1 1	Maximum speed forward
0xBE	1 0 1 1 1 1 1 0	
...	...	
0x81	1 0 0 0 0 0 0 1	Minimum speed forward
0x80	1 0 0 0 0 0 0 0	stopped
0xC0	1 1 0 0 0 0 0 0	stopped
0xC1	1 1 0 0 0 0 0 1	Minimum speed reverse
...	...	
0xFE	1 1 1 1 1 1 1 0	
0xFF	1 1 1 1 1 1 1 1	Maximum speed reverse

6.2.4.2 MOTOR_STEP Motor actions which increment or decrement the motor speed can specify the magnitude of the change with the MOTOR_STEP parameter. It is defined as follows:

MOTOR_STEP	Value
0x0	default +/-1 step (highest resolution)
0x1	1% (100 speed steps)
0x2	2% (50 speed steps)
0x3	3% (33 speed steps)
0x4	5% (20 speed steps)
0x5	6% (16 speed steps)
0x6	10% (10 speed steps)
0x7	20% (5 speed steps)
0x8	25% (4 speed steps)
0x9	33% (3 speed steps)
0xA	Lego compatible 7 step
0xB	15 deg (servo motor position increment)

The percentage change in speed is specified as an increment equal to that percentage of full speed.

6.2.4.3 MOTOR_PERIOD The MOTOR_PERIOD parameter specifies the time period for oscillating motor actions. This parameter is defined as follows:



For motor actions which have both an on and off interval, they can be specified individually. The definition of the motor period for both the ON and OFF period are defined as follows:

Value	Definition	Value	Definition
0x00	0.25 sec	0x08	3.0 sec
0x01	0.5 sec	0x09	4.0 sec
0x02	0.75 sec	0x0A	5.0 sec
0x03	1.0 sec	0x0B	10 sec
0x04	1.25 sec	0x0C	15 sec
0x05	1.5 sec	0x0D	20 sec
0x06	2.0 sec	0x0E	30 sec
0x07	2.5 sec	0x0F	60 sec

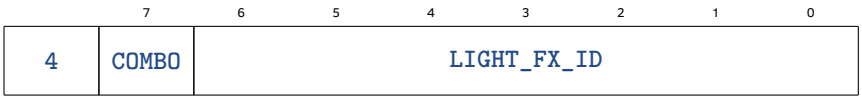
6.2.4.4 DURATION The DURATION parameter specifies a fixed time interval as follows:

Value	Definition	Value	Definition
0x0	0.5 sec	0x8	15 sec
0x1	1.0 sec	0x9	20 sec
0x2	1.5 sec	0xA	30 sec
0x3	2.0 sec	0xB	45 sec
0x4	3.0 sec	0xC	60 sec
0x5	4.0 sec	0xD	90 sec
0x6	5.0 sec	0xE	2 min
0x7	10 sec	0xF	5 min

6.2.4.5 MOTOR_POS The MOTOR_POS parameter specifies the angular position of a LEGO Power Functions servo motor. This parameter offers a convenient method of specifying the servo positon rather than a corresponding voltage or speed.

Value	Definition	Value	Definition
0x0	-90 deg	0x8	30 deg
0x1	-75 deg	0x9	45 deg
0x2	-60 deg	0xA	60 deg
0x3	-45 deg	0xB	75 deg
0x4	-30 deg	0xC	90 deg
0x5	-15 deg		
0x6	0 deg		
0x7	15 deg		

6.2.5 LIGHT_FX_ID



Light F/X actions are specified with an ID code which determines the action. There are two main types of light f/x: single and combination. Single light actions are applied to individually assigned lighting outputs (specified with the LIGHT_OUTPUT_MASK bytes). Combination light f/x are coordinated to drive an entire group of light outputs in a specific pattern. These combo light effects are applied to designated light output channels and override their current state when activated.

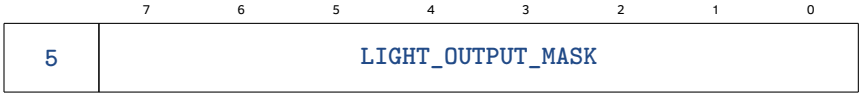
The COMBO bit <7> of the LIGHT_FX_ID byte specifies if the light f/x is a single or combo light action when set to 0 or 1 respectively. Based on the state of COMBO bit, the LIGHT_FX_ID is interpreted differently.

6.2.6 LIGHT_FX_ID Single Light Actions

When the COMBO bit is zero, then the LIGHT_FX_ID field is defined as follows:

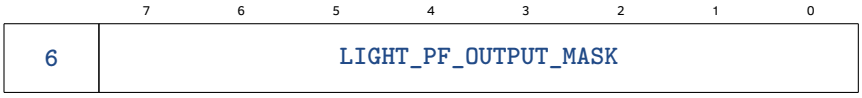
ID	MNEMONIC	Description
0x01	LIGHTFX_ON_OFF_TOGGLE	Toggle light output on/off
0x02	LIGHTFX_INC_BRIGHTNESS	Increase brightness one step
0x03	LIGHTFX_DEC_BRIGHTNESS	Decrease brightness one step
0x04	LIGHTFX_SET_BRIGHTNESS	Set brightness to specified level
0x05	LIGHTFX_FLASH_50_POS	50% duty cycle flasher (pos phase)
0x06	LIGHTFX_FLASH_50_NEG	50% duty cycle flasher (neg phase)
0x07	LIGHTFX_STROBE_POS	strobe light flasher (pos phase)
0x08	LIGHTFX_STROBE_NEG	strobe light flasher (neg phase)
0x09	LIGHTFX_GYRALITE_POS	fading MARS/Gyralite flasher (pos phase)
0x0A	LIGHTFX_GYRALITE_NEG	fading MARS/Gyralite flasher (neg phase)
0x0B	LIGHTFX_FLICKER	random flickering light
0x0C	LIGHTFX_RANDOM_BLINK	random blinking light
0x0D	LIGHTFX_PHOTON_TORPEDO	photon torpedo effect
0x0E	LIGHTFX_LASER_PULSE	shooting laser effect
0x0F	LIGHTFX_SCIFI_ENGINE_GLOW	glowing/pulsating engine glow effect
0x10	LIGHTFX_LIGHTHOUSE	rotating lighthouse effect
0x11	LIGHTFX_BROKEN_LIGHT	flickering faulty light effect
0x12	LIGHTFX_STATUS_INDICATOR	a status indicator of PFX events and status
0x13	LIGHTFX_SOUND_MODULATED	sound modulated light intensity
0x14	LIGHTFX_MOTOR_MODULATED	motor speed modulated light intensity

6.2.7 LIGHT_OUTPUT_MASK

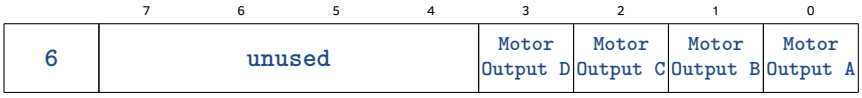


The selected light f/x can be applied to any combination of the dedicated light output ports. This is configured by the LIGHT_OUTPUT_MASK byte where a logic 1 in each bit corresponds to selected light output port, e.g. a LIGHT_OUTPUT_MASK<7:0>=0xC5 means that the light f/x will be applied to light output ports 8,7,3, and 1.

6.2.8 LIGHT_PF_OUTPUT_MASK



In addition to single light actions being applicable to the 8 dedicated light output ports, they can also be applied to the Power Functions motor output connectors. This is to support the use of the Lego brand 8870 dual LED lights with all of the sophisticated light f/x offered by the PFx Brick. Therefore, a single light action can be performed on up to 12 light output ports simultaneously. The LIGHT_PF_OUTPUT_MASK byte specifies which Power Functions output connectors are used as follows:



Note, that if a conflicting event/action simultaneously commands a motor output and a light action on the same Power Functions motor output port, the motor action will take priority and the light f/x action will be ignored.

6.2.9 LIGHT_PARAMx Single Light Actions

	7 6 5 4 3 2 1 0
7	LIGHT_PARAM1
8	LIGHT_PARAM2
9	LIGHT_PARAM3
10	LIGHT_PARAM4
11	LIGHT_PARAM5

Each of the single light actions defined by LIGHT_FX_ID field may have up to 5 additional parameter bytes to modify the behaviour of the light f/x. Currently, only the first 4 bytes have corresponding parameter assignments with byte 5 reserved for future use. The LIGHT_PARAM1, LIGHT_PARAM2, and LIGHT_PARAM3 bytes are assigned as follows to single action light f/x:

ID	MNEMONIC	LIGHT_PARAM1	LIGHT_PARAM2	LIGHT_PARAM3
0x01	LIGHTFX_ON_OFF_TOGGLE	DIR_OPTION	FADE_TIME	FLICKER_ON
0x02	LIGHTFX_INC_BRIGHTNESS			
0x03	LIGHTFX_DEC_BRIGHTNESS			
0x04	LIGHTFX_SET_BRIGHTNESS	BRIGHTNESS		
0x05	LIGHTFX_FLASH_50_POS	PERIOD	FADE_FACTOR	
0x06	LIGHTFX_FLASH_50_NEG	PERIOD	FADE_FACTOR	
0x07	LIGHTFX_STROBE_POS	PERIOD	DUTY_CYCLE	BURST_COUNT
0x08	LIGHTFX_STROBE_NEG	PERIOD	DUTY_CYCLE	BURST_COUNT
0x09	LIGHTFX_GYRALITE_POS	PERIOD	FADE_FACTOR	
0x0A	LIGHTFX_GYRALITE_NEG	PERIOD	FADE_FACTOR	
0x0B	LIGHTFX_FLICKER	PERIOD2	FADE_FACTOR	
0x0C	LIGHTFX_RANDOM_BLINK	PERIOD2	FADE_FACTOR	
0x0D	LIGHTFX_PHOTON_TORPEDO	PERIOD2		
0x0E	LIGHTFX_LASER_PULSE	PERIOD2		
0x0F	LIGHTFX_SCIFI_ENGINE_GLOW	PERIOD	FADE_FACTOR	
0x10	LIGHTFX_LIGHTHOUSE	PERIOD		
0x11	LIGHTFX_BROKEN_LIGHT	FAULT_RATE	FADE_TIME	FAULT_INTENSITY
0x12	LIGHTFX_STATUS_INDICATOR	SOURCE1	SOURCE2	INVERT
0x13	LIGHTFX_SOUND_MODULATED	FADE_TIME		INVERT
0x14	LIGHTFX_MOTOR_MODULATED	FADE_TIME	SOURCE2	INVERT

The LIGHT_PARAM4 parameter is used to qualify the tranisition behaviour of the light. Normally, an individual light action results in toggling the light output on or off. However, this can be qualified to assert the light output to either on or off rather than a toggle action. The LIGHT_PARAM4 byte is defined as follows:

	7	6	5	4	3	2	1	0
10	LIGHT_PARAM4							
10	DURATION				Reserved		TRANSITION	

The TRANSITION parameter is defined as follows:

Value	Description
0x00	toggle light output
0x01	turn light ON
0x02	turn light OFF
0x03	turn light ON for a specified DURATION

The DURATION parameter specifies a fixed time interval as follows:

Value	Definition	Value	Definition
0x0	0.5 sec	0x8	15 sec
0x1	1.0 sec	0x9	20 sec
0x2	1.5 sec	0xA	30 sec
0x3	2.0 sec	0xB	45 sec
0x4	3.0 sec	0xD	60 sec
0x5	4.0 sec	0xD	90 sec
0x6	5.0 sec	0xE	2 min
0x7	10 sec	0xF	5 min

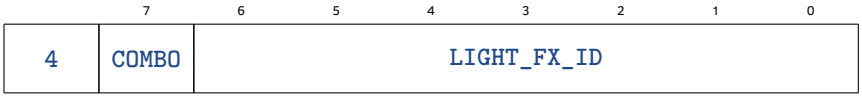
Suggested Default Parameter Values

The table below shows some suggested default values for a host application for each light f/x.

ID	MNEMONIC	LIGHT_PARAM1	LIGHT_PARAM2	LIGHT_PARAM3
0x01	LIGHTFX_ON_OFF_TOGGLE	0x00 : None	0x00 : No Fade	0x00 : No flicker
0x02	LIGHTFX_INC_BRIGHTNESS			
0x03	LIGHTFX_DEC_BRIGHTNESS			
0x04	LIGHTFX_SET_BRIGHTNESS	0x7F		
0x05	LIGHTFX_FLASH_50_POS	0x04 : 1 sec.	0x03 : 10%	
0x06	LIGHTFX_FLASH_50_NEG	0x04 : 1 sec.	0x03 : 10%	
0x07	LIGHTFX_STROBE_POS	0x04 : 1 sec.	0x04 : 15%	0x01 : 2 pulses
0x08	LIGHTFX_STROBE_NEG	0x04 : 1 sec.	0x04 : 15%	0x01 : 2 pulses
0x09	LIGHTFX_GYRALITE_POS	0x04 : 1 sec.	0x09 : 50%	
0x0A	LIGHTFX_GYRALITE_NEG	0x04 : 1 sec.	0x09 : 50%	
0x0B	LIGHTFX_FLICKER	0x01 : 0.1 sec.	0x06 : 25%	
0x0C	LIGHTFX_RANDOM_BLINK	0x02 : 0.2 sec.	0x03 : 10%	
0x0D	LIGHTFX_PHOTON_TORPEDO	0x0A : 1 sec.		
0x0E	LIGHTFX_LASER_PULSE	0x01 : 0.1 sec.		
0x0F	LIGHTFX_SCIFI_ENGINE_GLOW	0x08 : 2 sec.	0x09 : 50%	
0x10	LIGHTFX_LIGHTHOUSE	0x0A : 3 sec.		
0x11	LIGHTFX_BROKEN_LIGHT	0x02 : Often	0x00 : None	0x02 : Severe
0x12	LIGHTFX_STATUS_INDICATOR	0x01	0x00	0x00
0x13	LIGHTFX_SOUND_MODULATED	0x03 : 10%		
0x14	LIGHTFX_MOTOR_MODULATED	0x03 : 10%	0x00 : None	0x00 : not inverted

The suggested default value for LIGHT_PARAM4 is 0x00, i.e. toggle light output on/off.

6.2.10 LIGHT_FX_ID Combination Light Actions



When the COMBO bit is set, then the LIGHT_FX_ID corresponds to a combination light action specified in the table below. Unlike masked combinations of single light actions, these effects are coordinated to drive all 8x light outputs in a specific pattern. These combo light effects are applied to all 8x light f/x channels and override their current state when activated.

ID	MNEMONIC	Description
0x01	COMBOFX_LINEAR_SWEEP	Linear sweep of sequential lights
0x02	COMBOFX_BARGRAPH_SWEEP	Linear bargraph sweep
0x03	COMBOFX_KNIGHT_RIDER	Knight rider back-forth scanner
0x04	COMBOFX_EMCY_TWINSONIC	Emergency vehicle with twinsonic lightbars
0x05	COMBOFX_EMCY_WHELEN	Emergency vehicle with Whelen lightbars
0x06	COMBOFX_TIMES_SQUARE	Constantly changing patterns of sweeping lights
0x07	COMBOFX_NOISE	Random patterns
0x08	COMBOFX_TWINKLING_STARS	Simulated twinkling star field effect
0x09	COMBOFX_TRAFFIC_LIGHTS	Traffic light sequence including pedestrian crossing
0x0A	COMBOFX_SOUND_BAR	A bargraph modulated by sound playback
0x0B	COMBOFX_ALTERNATE_FLASH	A pair of lights which flash in opposite phases
0x0C	COMBOFX_LAVA_LAMP	A soft fluid modulated light effect
0x0D	COMBOFX_LASER_CANNON	A one-shot sweeping light effect
0x0E	COMBOFX_DRAGSTER	Dragster starter signal lights
0x0F	COMBOFX_RUNWAY	Airport runway approach lights
0x10	COMBOFX_FORMULA1	Formula 1 style starter lights

6.2.11 Combination Light F/X Notes

6.2.11.1 Sound Bargraph The sound bargraph light f/x animates a bargraph type display in response to any audio playback activity. The bargraph deflects at a level proportional to the instantaneous audio level. The bargraph style as well as the number of lights and peristence can be configured using parameters `BAR_STYLE`, `SIZE`, and `FADE_FACTOR`.

6.2.11.2 Traffic Lights Traffic lights for a typical four way intersection can be simulated with the traffic lights combo light f/x. The two opposing flows of traffic are designated North/South and East/West and each have a dedicated group of Red, Yellow, and Green light aspects. Additionally, the North/South flow has two optional light outputs for a pedestrian crossing indicator with "Walk" and "Don't Walk" aspects. The assignment of light output channels to the corresponding light aspects is as follows:

Light Output							
1	2	3	4	5	6	7	8
R	Y	G	R	Y	G	Don't Walk	Walk
North/South			East/West			North/South Ped Crossing	

6.2.11.3 Emergency Flashers The combo light f/x which are used to simulate flashers on emergency vehicles (e.g. police, fire, ambulance, etc.) enable builders to configure light outputs to match a wide variety of emergency vehicles used around the world and from different eras. A key feature of emergency vehicle flashers is the roof mounted lighting; implemented either as discrete lights or more commonly mounted into a light bar structure on the roof. In addition to the roof/lightbar flashers are auxilary flashing lights. These auxilary lights vary widely in terms of quantity and location among all emergency vehicles. Examples include side mounted flashers, radiator grille flashers, headlamp cluster flashers, etc. Auxilary flashers are often synchronized with one or more of the lightbar flashers and may or may not have the same flashing pattern. The PFx Brick provides a variety of functional flashing light outputs for all emergency flasher types in order to match a wide variety of prototypical emergency vehicles. The builder does not have to use every light output and may chose any combination which best suits their model. The emergency flashers use 6 of 8 light output ports, leaving the 7th and 8th port available for another use, e.g. headlamps.

For all emergency flasher applications, the light outputs are defined the same way. They are as follows:

Light Output					
1	2	3	4	5	6
Light bar				Auxilary Flashers	
Left		Right		1x flash	
outer	inner	inner	outer	left	right

6.2.11.4 Light Bar The lightbar or roof mounted lights consist of a group of 4 lights which flash in variety of different styles. Often, these lights will be co-packaged into a roof mounted light bar. Two lights are intended for the left side of the vehicle and another pair is intended for the right side. Each left/right pair can have an inner and outer light. This allows light flashing sequences to alternate

from left to right or from inside to outside depending on the style. For more simple applications, one of each of the left and right pairs can be used, e.g. just the outer left/right pair.

Two very common types of lightbar flashers are the so-called “Twinsonic” and “Whelen” style lightbars. These are named after the trade-marked products of Federal Signal and Whelen Engineering respectively; manufacturers of emergency vehicle lighting products. These style names are intended to be representative and not exact copies of any particular lighting product. The “Twinsonic” style light bar physically consisted of rotating mirrors around a light source and were common in older or heritage emergency vehicles. The rotating light effect is simulated with periodically variable brightness and has a “softer” flashing effect. The “Whelen” style lightbar is designed to simulate the flashing effects of modern and contemporary LED strobe-type emergency flashers. These light bars have many different strobe-like patterns and sequences. The PFx Brick includes most of the typical sequences available from this style of emergency flasher.

6.2.11.5 Auxilary Flashers Many emergency vehicles incorporate additional flashing lights to those mounted on the roof. These can consist of flashers which duplicate the flashing sequence from the light bar or flash periodically synchronized with the alternating effect of the lightbar. The PFx Brick provides auxilary flasher outputs in order to connect lights which best represent the flashing light configuration of a particular vehicle.

The left/right auxilary 1x flashers flash periodically at the specificied rate alternating from left to right. The 1x and auxilary flashers are simple periodic flashers and do not exhibit the complicated flash sequences of the light bar. They are however synchronized with the light bar flash rate.

6.2.12 LIGHT_PARAMx Combination Light Actions

	7	6	5	4	3	2	1	0
7	LIGHT_PARAM1							
8	LIGHT_PARAM2							
9	LIGHT_PARAM3							
10	LIGHT_PARAM4							
11	LIGHT_PARAM5							

Each of the combination light actions defined by LIGHT_FX_ID field may have up to 5 additional parameter bytes to modify the behaviour of the light f/x. Most of the combo light f/x use 2 to 4 parameter bytes with the remaining bytes reserved for future use. The LIGHT_PARAM1 - LIGHT_PARAM4 bytes are assigned as follows to combination light f/x:

ID	MNEMONIC	LIGHT_PARAM1	LIGHT_PARAM2	LIGHT_PARAM3
0x01	COMBOFX_LINEAR_SWEEP	PERIOD	FADE_FACTOR	SIZE
		LIGHT_PARAM4		
		SWEEP_STYLE		
0x02	COMBOFX_BARGRAPH_SWEEP	PERIOD	FADE_FACTOR	SIZE
		LIGHT_PARAM4		
		SWEEP_STYLE		
0x03	COMBOFX_KNIGHT_RIDER	PERIOD	FADE_FACTOR	SIZE
0x04	COMBOFX_EMCY_TWINSONIC	TWIN_STYLE	SEQ	FLASH_RATE
0x05	COMBOFX_EMCY_WHELEN	WHELEN_STYLE	SEQ	FLASH_RATE
0x06	COMBOFX_TIMES_SQUARE	PERIOD2	FADE_FACTOR	
0x07	COMBOFX_NOISE	PERIOD2	FADE_FACTOR	
0x08	COMBOFX_TWINKLING_STARS	PERIOD	FADE_FACTOR	
0x09	COMBOFX_TRAFFIC_LIGHTS	TRAFFIC_STYLE	FADE_FACTOR	SEQ_TIME
0x0A	COMBOFX_SOUND_BAR	BAR_STYLE	FADE_FACTOR	SIZE
0x0B	COMBOFX_ALTERNATE_FLASH	PERIOD	FADE_FACTOR	DUTY_CYCLE
		LIGHT_PARAM4	LIGHT_PARAM5	
		OUT_MASK	TRANSITION	
0x0C	COMBOFX_LAVA_LAMP	PERIOD	SIZE	
0x0D	COMBOFX_LASER_CANON	FLASH_RATE	FADE_FACTOR	SIZE
		LIGHT_PARAM4		
		SWEEP_STYLE		
0x0E	COMBOFX_DRAGSTER	DRAGSTER_STYLE	FADE_FACTOR	
0x0F	COMBOFX_RUNWAY	RUNWAY_RATE	FADE_FACTOR	RUNWAY_BRIGHT
0x10	COMBOFX_FORMULA1	F1_STYLE	FADE_FACTOR	FLASH_RATE

Suggested Default Parameter Values

The table below shows some suggested default values for a host application for each light f/x.

ID	MNEMONIC	LIGHT_PARAM1	LIGHT_PARAM2	LIGHT_PARAM3
0x01	COMBOFX_LINEAR_SWEEP	0x02 : 0.5 sec.	0x06 : 25%	0x00 : 8 lights
		LIGHT_PARAM4		
		0x01 : R to L		
0x02	COMBOFX_BARGRAPH_SWEEP	0x04 : 1.0 sec.	0x03 : 10%	0x00 : 8 lights
		LIGHT_PARAM4		
		0x01 : R to L		
0x03	COMBOFX_KNIGHT_RIDER	0x06 : 1.5 sec.	0x06 : 25%	0x00 : 8 lights
0x04	COMBOFX_EMCY_TWINSONIC	0x02 : Aero	0x01 : L/R	0x02 : Fast
0x05	COMBOFX_EMCY_WHELEN	0x0A : Random	0x02 : In/Out	0x02 : Fast
0x06	COMBOFX_TIMES_SQUARE	0x01 : 0.1 sec.	0x0A : 75%	
0x07	COMBOFX_NOISE	0x01 : 0.1 sec.	0x09 : 50%	
0x08	COMBOFX_TWINKLING_STARS	0x08 : 2 sec.	0x0F : 400%	
0x09	COMBOFX_TRAFFIC_LIGHTS	0x04 : Std w/Xing	0x06 : 25%	0x01 : Med
0x0A	COMBOFX_SOUND_BAR	0x00 : L to R	0x06 : 25%	0x00 : 8 lights
0x0B	COMBOFX_ALTERNATE_FLASH	0x04 : 1 sec.	0x09 : 50%	0x06 : 25%
		LIGHT_PARAM4	LIGHT_PARAM5	
		0x0F	0x00 Toggle	
0x0C	COMBOFX_LAVA_LAMP	0x04 : 1 sec.	0x00 : 8 lights	
0x0D	COMBOFX_LASER_CANON	0x03 : Very Fast	0x06 : 25%	0x04 : 4 lights
		LIGHT_PARAM4		
		0x01 : R to L		
0x0E	COMBOFX_DRAGSTER	0x00 Starter	0x03 10%	
0x0F	COMBOFX_RUNWAY	0x02 Med	0x03 10%	0x00 Maximum
0x10	COMBOFX_FORMULA1	0x00 Race Start	0x03 10%	0x01 Med

6.2.13 LIGHT_PARAMx Definitions

This section describes all of the named parameters occupying the LIGHT_PARAMx event action bytes. Many of the parameters are shared among both single and combination light f/x.

6.2.13.1 DIR_OPTION The DIR_OPTION parameter qualifies the illumination of individual lighting events based on motor direction. This can be used for directional head and tail lamps on a motor powered vehicle for example.

Value	Description
0x00	No directional behaviour
0x01	Lights illuminate if Motor A is FWD
0x02	Lights illuminate if Motor A is REV
0x03	Lights illuminate if Motor B is FWD
0x04	Lights illuminate if Motor B is REV
0x05	Lights illuminate if Motor C is FWD
0x06	Lights illuminate if Motor C is REV
0x07	Lights illuminate if Motor D is FWD
0x08	Lights illuminate if Motor D is REV
0x09	Odd lights illuminate if Motor A is FWD, even lights if REV
0x0A	Odd lights illuminate if Motor B is FWD, even lights if REV
0x0B	Odd lights illuminate if Motor C is FWD, even lights if REV
0x0C	Odd lights illuminate if Motor D is FWD, even lights if REV
0x0D	Odd lights illuminate if Motor A is REV, even lights if FWD
0x0E	Odd lights illuminate if Motor B is REV, even lights if FWD
0x0F	Odd lights illuminate if Motor C is REV, even lights if FWD
0x10	Odd lights illuminate if Motor D is REV, even lights if FWD

6.2.13.2 FLICKER_ON The FLICKER_ON parameter specifies whether a light should flicker during its transition from off to on. Any non-zero value will enable this feature.

6.2.13.3 OUT_MASK The OUT_MASK parameter corresponds to an light output mask with bits 7-0 corresponding to light output ports 8-1 respectively. A 1 in a bit position indicates that the corresponding light output port should be used/active.

6.2.13.4 FADE_TIME The FADE_TIME parameter specifies the absolute duration of intensity fading when the light transitions to a different intensity levels.

Value	Definition	Value	Definition
0x00	No Fade	0x08	1.0 sec
0x01	50 ms	0x09	1.5 sec
0x02	0.1 sec	0x0A	2.0 sec
0x03	0.2 sec	0x0B	2.5 sec
0x04	0.4 sec	0x0C	3.0 sec
0x05	0.5 sec	0x0D	4.0 sec
0x06	0.6 sec	0x0E	5.0 sec
0x07	0.8 sec	0x0F	10.0 sec

6.2.13.5 FADE_FACTOR The FADE_FACTOR parameter specifies the duration (relative to the period of the light f/x) of intensity fading when the light transitions to a different intensity levels.

Value	Definition	Value	Definition
0x00	No Fade	0x08	40 %
0x01	1 %	0x09	50 %
0x02	5 %	0x0A	75 %
0x03	10 %	0x0B	90 %
0x04	15 %	0x0C	100 %
0x05	20 %	0x0D	150 %
0x06	25 %	0x0E	200 %
0x07	30 %	0x0F	400 %

6.2.13.6 PERIOD The PERIOD parameter specifies repeating period for many light f/x.

Value	Definition	Value	Definition
0x00	0.1 sec	0x08	2.0 sec
0x01	0.25 sec	0x09	2.5 sec
0x02	0.5 sec	0x0A	3.0 sec
0x03	0.75 sec	0x0B	4.0 sec
0x04	1.0 sec	0x0C	5.0 sec
0x05	1.25 sec	0x0D	8.0 sec
0x06	1.5 sec	0x0E	10.0 sec
0x07	1.75 sec	0x0F	20.0 sec

6.2.13.7 PERIOD2 The PERIOD2 parameter specifies repeating period for many light f/x.

Value	Definition	Value	Definition
0x00	0.05 sec	0x08	0.8 sec
0x01	0.1 sec	0x09	0.9 sec
0x02	0.2 sec	0x0A	1.0 sec
0x03	0.3 sec	0x0B	1.25 sec
0x04	0.4 sec	0x0C	1.5 sec
0x05	0.5 sec	0x0D	1.75 sec
0x06	0.6 sec	0x0E	2.0 sec
0x07	0.7 sec	0x0F	3.0 sec

6.2.13.8 TRANSITION The TRANSITION parameter used with the alternating flash effect defines the transition after the active (flashing) state. It is defined as follows:

Value	Description
0x00	toggle light output
0x01	transition to always ON
0x02	transition to OFF

6.2.13.9 DUTY_CYCLE The DUTY_CYCLE parameter specifies ratio of On/Off intervals for several periodic light f/x.

Value	Definition	Value	Definition
0x00	1%	0x0A	60%
0x01	2%	0x0B	70%
0x02	5%	0x0C	75%
0x03	10%	0x0D	80%
0x04	15%	0x0E	85%
0x05	20%	0x0F	90%
0x06	25%	0x10	95%
0x07	30%	0x11	98%
0x08	40%	0x12	99%
0x09	50%		

6.2.13.10 BURST_COUNT The BURST_COUNT parameter specifies how many consective strobe intervals a LIGHTFX_STROBE_POS/NEG light f/x has. Generally, the strobe intervals are much shorter than the overall period of the light f/x and are specified with the DUTY_CYCLE parameter.

Value	Description
0x00	1 strobe pulse
0x01	2 strobe pulses
0x02	3 strobe pulses
0x03	4 strobe pulses

6.2.13.11 SIZE The SIZE parameter restricts the number of light outputs used for combo light f/x. Most combo light f/x use up to all 8 light output channels; however, some light f/x can be scaled to use less light channels. Restricting the size of the combo action makes the remaining light channels available for other light f/x actions.

Value	Description
0x00	8 lights
0x01	7 lights
0x02	6 lights
0x03	5 lights
0x04	4 lights

6.2.13.12 BAR_STYLE The BAR_STYLE parameter determines the modulation pattern of combo light f/x such as the sound bar.

Value	Description
0x01	Left to Right bar graph
0x02	Right to Left bar graph
0x03	In to Out symmetric bar graph
0x04	Out to In symmetric bar graph

6.2.13.13 TWINSONIC_STYLE The TWINSONIC_STYLE parameter determines the modulation pattern of the Twinsonic emergency flasher combo light f/x.

Value	Description
0x00	Single
0x01	Dual
0x02	Aero
0x03	Combo

6.2.13.14 WHELEN_STYLE The WHELEN_STYLE parameter determines the modulation pattern of the Whelen light bar emergency flasher combo light f/x.

Value	Description
0x0	Signal Alert
0x1	Signal Alert Steady
0x2	Comet Flash
0x3	Action Flash 50
0x4	Action Flash 150
0x5	Modu Flash
0x6	Single Flash
0x7	Double Flash
0x8	Triple Flash
0x9	Warning
0xA	Random

6.2.13.15 SWEEP_STYLE The SWEEP_STYLE parameter determines the modulation pattern of combo light f/x such as linear sweep and bar graph.

Value	Description
0x00	Left to Right pattern
0x01	Right to Left pattern

6.2.13.16 TRAFFIC_STYLE The TRAFFIC_STYLE parameter determines the type of traffic light sequence to simulate.

Value	Description
0x00	Standard
0x01	Standard with flashing green
0x02	European
0x03	Flashing red (NS), flashing yellow (EW)
0x04	Standard with pedestrian crossing
0x05	Standard with flashing green and pedestrian crossing
0x06	European with pedestrian crossing
0x07	Flashing red (EW), flashing yellow (NS)
0x08	International
0x09	International with pedestrian crossing
0x08	International 2
0x09	International 2 with pedestrian crossing

6.2.13.17 SEQ_TIME The SEQ_TIME parameter determines the length of traffic light sequence.

Value	Description
0x00	Slow (60 sec)
0x01	Medium (45 sec)
0x02	Fast (30 sec)
0x03	Very Fast (20 sec)

6.2.13.18 SEQ The SEQ parameter determines how the flashing pattern is sequenced on emergency flasher light bars, e.g. alternating left and right, alternating from inside to outside, etc.

Value	Description
0x00	Solid
0x01	Left/Right
0x02	In/Out

6.2.13.19 FLASH_RATE The FLASH_RATE parameter determines flashing rate of emergency flashers.

Value	Description
0x00	Slow (60 fpm)
0x01	Medium (90 fpm)
0x02	Fast (120 fpm)
0x03	Very Fast (150 fpm)

6.2.13.20 FAULT_RATE The FAULT_RATE parameter determines the approximate probability of the broken light flickering.

Value	Description
0x00	Rare (5%)
0x01	Occasionally (10%)
0x02	Often (25%)
0x03	Very Often (50%)

6.2.13.21 FAULT_INTENSITY The FAULT_INTENSITY parameter determines the approximate relative change of intensity of the broken light flickering.

Value	Description
0x00	Subtle
0x01	Moderate
0x02	Severe
0x03	Maximum

6.2.13.22 SOURCE1 The SOURCE1 parameter specifies a combination of internal PFx Brick events which can trigger a light channel. Each of the values listed can be logically OR-ed together to indicate multiple items on one light channel.

Value	Description
0x01	USB connected
0x02	USB activity
0x04	IR activity
0x08	IR lockout active
0x10	Audio playback active
0x20	BLE connected
0x40	BLE activity
0x80	Flash File System activity

6.2.13.23 SOURCE2 The SOURCE2 parameter specifies a combination of motor channel states which can trigger a light channel. The indication is only active when the motor channel is operating at a speed that is not zero. Each of the values listed can be logically OR-ed together to indicate multiple items on one light channel.

Value	Description
0x01	Motor channel A forward
0x02	Motor channel A reverse
0x04	Motor channel B forward
0x08	Motor channel B reverse
0x10	Motor channel C forward
0x20	Motor channel C reverse
0x40	Motor channel D forward / touchLAB Button State
0x80	Motor channel D reverse / Gated Motor Audio Playback Trigger

6.2.13.24 INVERT The INVERT parameter is used to specify whether the light channel output is inverted, i.e. an active state is shown with the light off. Normally, an active state is shown with the light on. When INVERT is zero, the indicator is normal, i.e. active=on. When set to a non-zero value, the indicator is inverted, i.e. active=off.

6.2.13.25 BRIGHTNESS A numeric value specifying light intensity. The valid range is 0 to 255 corresponding to minimum and maximum brightness respectively.

6.2.13.26 `DRAGSTER_STYLE` The dragster starting lights can operate in one of the 3 folloing styles:

Value	Description
0x00	Standard countdown to green
0x01	Pro countdown to green 0.5 sec
0x02	Pro countdown to green 0.4 sec

6.2.13.27 `F1_STYLE` The Formula 1 combo light effects can operate in a variety of styles which correspond to the different operational phases of a typical F1 race. The F1 styles are defined as follows:

Value	Description
0x00	Race start countdown
0x01	Training countdown
0x02	Race break / caution
0x03	Training start
0x04	Training break
0x05	Training end

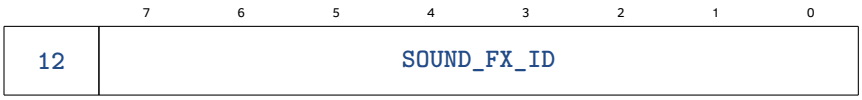
6.2.13.28 `RUNWAY_RATE` The runway approach flasher lights can operate with flashing rates defined as follows:

Value	Description
0x00	Steady - no flashing
0x01	Slow
0x02	Med
0x03	Fast

6.2.13.29 `RUNWAY_BRIGHT` The runway approach lights illuminate with a static brightness level under the animating flashing effect. The static brightness level is defined by this parameter as follows:

Value	Description
0x00	Maximum
0x01	Med
0x02	Low
0x03	Minimum

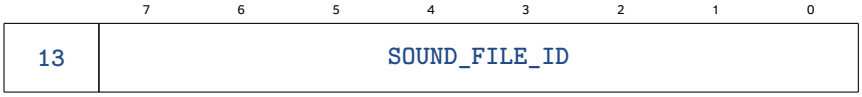
6.2.14 SOUND_FX_ID



Sound effects are actions to playback a specific sound "file" stored in flash memory. Sounds are stored in flash memory and are pre-loaded by the host PFX Application. Polyphonic mixing of sounds is the default behaviour so that sound f/x can be combined realistically. The SOUND_FX_ID encodes the sound f/x actions as follows:

ID	MNEMONIC	Description
0x00	SOUNDFX_NONE	No audio effect
0x01	SOUNDFX_INC_VOLUME	Increase master volume one step
0x02	SOUNDFX_DEC_VOLUME	Decrease master volume one step
0x03	SOUNDFX_SET_VOLUME	Set volume to a specific step
0x04	SOUNDFX_PLAY_ONCE	Play sound file one time
0x05	SOUNDFX_PLAY_CONTINUOUS	Play sound file continuously (effect can be toggled)
0x06	SOUNDFX_PLAY_NTIMES	Play sound file a specified number of times
0x07	SOUNDFX_PLAY_DURATION	Loop sound file playback for a specified duration
0x08	SOUNDFX_PLAY_PITCHBEND_MOTOR	Play sound file continuous; modulate pitch as a function of motor speed
0x09	SOUNDFX_PLAY_GATED_MOTOR	Play sound file; then silence at a rate proportional to motor speed (e.g. for "chuffing" sound)
0x0A	SOUNDFX_PLAY_AM_MOTOR	Play sound file continuous; modulate volume as a function of motor speed
0x0B	SOUNDFX_STOP	Stop playback of specified sound file
0x0C	SOUNDFX_PLAY_IDX_MOTOR	Play sound files automatically indexed by motor speed. Allows for realistic simulation of engine sounds stored in audio files.
0x0D	SOUNDFX_PLAY_RAND	Play sound file at randomly defined time intervals
0x0E	SOUNDFX_FILE_SEEK	Set the sound file pointer to an absolute time position in currently playing file.
0x0F	SOUNDFX_FILE_SCRUB	Set the sound file pointer with a relative time offset in the currently playing file.

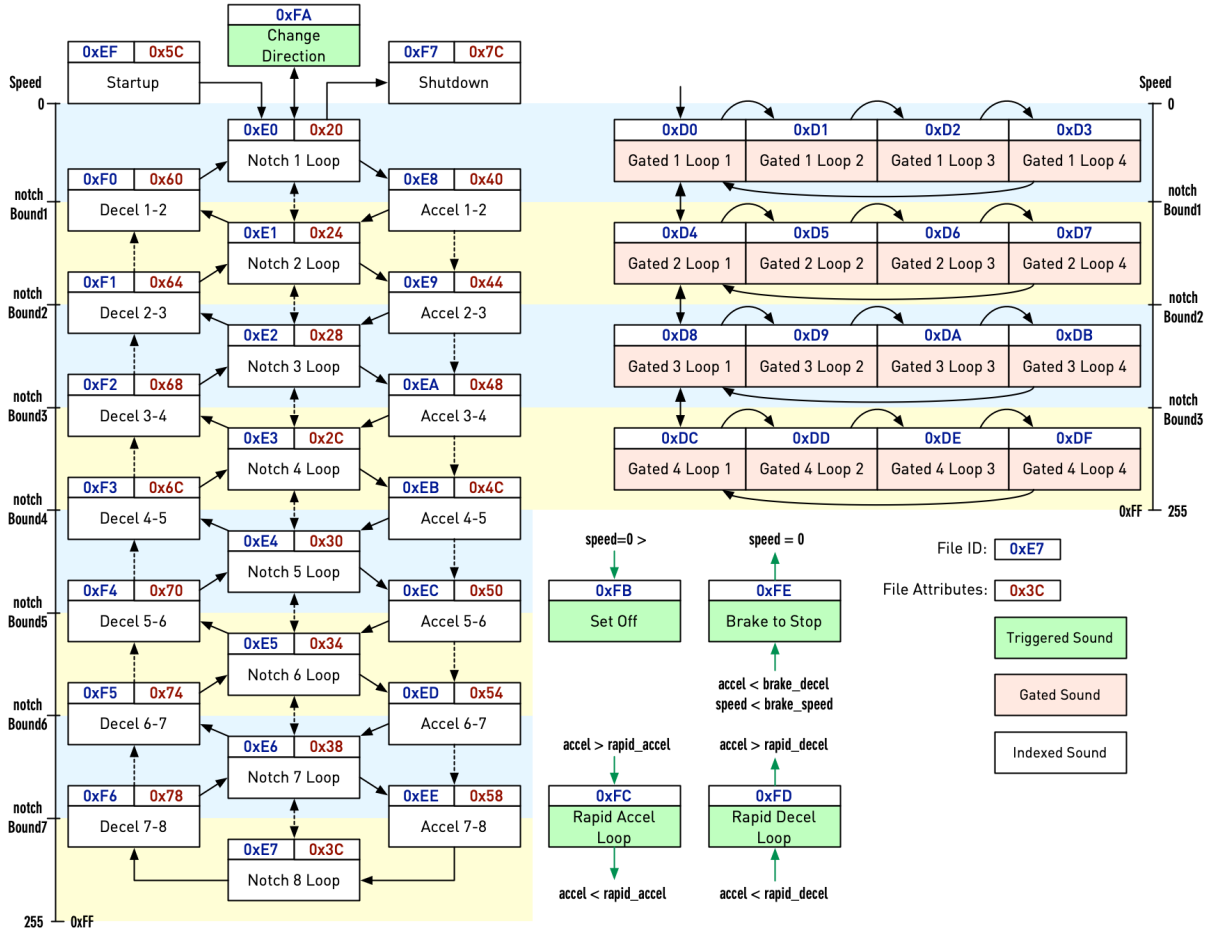
6.2.15 SOUND_FILE_ID



The SOUND_FILE_ID is the file ID of a sound file stored in the PFx Brick file system.

6.2.16 Sound F/X Notes

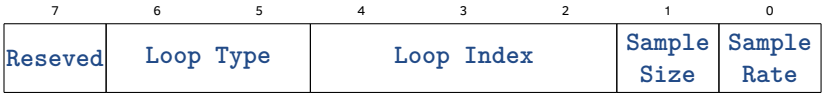
6.2.16.1 Indexed Motor/Engine Sounds (SOUNDFX_PLAY_IDX_MOTOR) One of the more sophisticated sound playback behaviours for the PFx Brick is the automatic playback of sound files to simulate engines, motors, prime-movers, etc. This requires specially prepared sound files which can be reliably looped and/or sequentially played without gaps and acoustically transition smoothly. The playback of sound files is automatically scheduled by the PFx Brick depending on the motor speed. A summary of this playback sound schedule can be found in the diagram below.



A motor sound will typically have different acoustic properties depending on the speed or load of the motor. For example, as a motor increases or decrease speed or rpm, its pitch will increase/decrease proportionally to its speed. In order to simulate the sound of the motor, the PFx Brick can loop up to 8 different sound file loops representing the sound of the motor at each speed or power level called “notches”. In the PFx Brick configuration, the number of power notches can be specified as well as the speed level between each notch. Details of this configuration can be found in the PFX_CMD_SET_CONFIG section.

For maximum fidelity, the sound of the motor transitioning between each power notch (accelerating and/or decelerating) can be represented with a dedicated sound file for each transition. Lastly, dedicated sound files for a motor startup and shutdown sound can also be specified.

In order to designate sound files stored on the PFx Brick for use with SOUNDFX_PLAY_IDX_MOTOR sound effect, the files have special attributes set in the file’s directory listing. In particular, the lower byte of the User Attributes field of the directory entry has special bits which tag the file as follows:



Using this scheme, the sound files that can be specified for motor speed indexed playback can be summarized as follows:

File	User Attributes[7:0]	Loop Type	Loop Index	Description
Gated 1 Loop	X001 00XX [0x10]	00	100	Gated playback file 1
Gated 2 Loop	X001 01XX [0x14]	00	101	Gated playback file 2
Gated 3 Loop	X001 10XX [0x18]	00	110	Gated playback file 3
Gated 4 Loop	X001 11XX [0x1C]	00	111	Gated playback file 4
Notch 1 Loop	X010 00XX [0x20]	01	000	Loop for minimum speed
Notch 2 Loop	X010 01XX [0x24]	01	001	Loop for speed notch 2
Notch 3 Loop	X010 10XX [0x28]	01	010	Loop for speed notch 3
Notch 4 Loop	X010 11XX [0x2C]	01	011	Loop for speed notch 4
Notch 5 Loop	X011 00XX [0x30]	01	100	Loop for speed notch 5
Notch 6 Loop	X011 01XX [0x34]	01	101	Loop for speed notch 6
Notch 7 Loop	X011 10XX [0x38]	01	110	Loop for speed notch 7
Notch 8 Loop	X011 11XX [0x3C]	01	111	Loop for speed notch 8
Accel 1-2	X100 00XX [0x40]	10	000	Sound transition from notch 1 to 2
Accel 2-3	X100 01XX [0x44]	10	001	Sound transition from notch 2 to 3
Accel 3-4	X100 10XX [0x48]	10	010	Sound transition from notch 3 to 4
Accel 4-5	X100 11XX [0x4C]	10	011	Sound transition from notch 4 to 5
Accel 5-6	X101 00XX [0x50]	10	100	Sound transition from notch 5 to 6
Accel 6-7	X101 01XX [0x54]	10	101	Sound transition from notch 6 to 7
Accel 7-8	X101 10XX [0x58]	10	110	Sound transition from notch 7 to 8
Startup	X101 11XX [0x5C]	10	111	Startup sound
Decel 2-1	X110 00XX [0x60]	11	000	Sound transition from notch 2 to 1
Decel 3-2	X110 01XX [0x64]	11	001	Sound transition from notch 3 to 2
Decel 4-3	X110 10XX [0x68]	11	010	Sound transition from notch 4 to 3
Decel 5-4	X110 11XX [0x6C]	11	011	Sound transition from notch 5 to 4
Decel 6-5	X111 00XX [0x70]	11	100	Sound transition from notch 6 to 5
Decel 7-6	X111 01XX [0x74]	11	101	Sound transition from notch 7 to 6
Decel 8-7	X111 10XX [0x78]	11	110	Sound transition from notch 8 to 7
Shutdown	X111 11XX [0x7C]	11	111	Shutdown sound

The process of preparing the PFx Brick for this sound effect can be summarized as follows:

1. Use the `PFX_CMD_SET_CONFIG` command to set the `Notch Count` for the desired number of fixed power notches to simulate (1 to 8)
2. Use the `PFX_CMD_SET_CONFIG` command to also set the speed boundaries between the power notches. These boundaries must be set in monotonically increasing order.
3. Load all of the desired audio files corresponding to the motor speed loops on to the PFx Brick file system.

4. Use the PFX_CMD_FILE_DIR command with a request type of 0x0A (set masked attributes with ID) to set attributes of each file. For example, to configure a file with ID 0x55 to be a loop file for notch 7, then the PFX_CMD_FILE_DIR command is as follows: 0x45 0x0A 0x55 0x00 0x38 0x00 0x7C The 0x007C is convenient mask so that only bits associated with the Loop Type and Loop Index are set, i.e. 0x0038.

6.2.17 Reserved File IDs

Automated sound playback modes such as SOUNDFX_PLAY_IDX_MOTOR and SOUNDFX_PLAY_GATED_MOTOR rely on the lower byte of the User Attributes field to designate files for a particular purpose, e.g. a startup sound, idle loop, etc. Since the capacity of the bits in the User Attributes field is becoming exhausted for this purpose, an additional method of designating files is implemented by using the actual File ID in the file system. A block of File ID values will be reserved to designate files for specialized audio playback modes. This method is optional and maintains backward compatibility with using the User Attributes field. In the case where both methods are used, the User Attributes will take priority. The table below shows the reserved File ID values and their corresponding purpose.

File	Reserved File ID	Description
Gated Notch 1 Loop 1	0xD0	Gated playback loop 1 in notch 1
Gated Notch 1 Loop 2	0xD1	Gated playback loop 2 in notch 1
Gated Notch 1 Loop 3	0xD2	Gated playback loop 3 in notch 1
Gated Notch 1 Loop 4	0xD3	Gated playback loop 4 in notch 1
Gated Notch 2 Loop 1	0xD4	Gated playback loop 1 in notch 2
Gated Notch 2 Loop 2	0xD5	Gated playback loop 2 in notch 2
Gated Notch 2 Loop 3	0xD6	Gated playback loop 3 in notch 2
Gated Notch 2 Loop 4	0xD7	Gated playback loop 4 in notch 2
Gated Notch 3 Loop 1	0xD8	Gated playback loop 1 in notch 3
Gated Notch 3 Loop 2	0xD9	Gated playback loop 2 in notch 3
Gated Notch 3 Loop 3	0xDA	Gated playback loop 3 in notch 3
Gated Notch 3 Loop 4	0xDB	Gated playback loop 4 in notch 3
Gated Notch 4 Loop 1	0xDC	Gated playback loop 1 in notch 4
Gated Notch 4 Loop 2	0xDD	Gated playback loop 2 in notch 4
Gated Notch 4 Loop 3	0xDE	Gated playback loop 3 in notch 4
Gated Notch 4 Loop 4	0xDF	Gated playback loop 4 in notch 4

File	Reserved File ID	Description
Notch 1 Loop	0xE0	Loop for minimum speed
Notch 2 Loop	0xE1	Loop for speed notch 2
Notch 3 Loop	0xE2	Loop for speed notch 3
Notch 4 Loop	0xE3	Loop for speed notch 4
Notch 5 Loop	0xE4	Loop for speed notch 5
Notch 6 Loop	0xE5	Loop for speed notch 6
Notch 7 Loop	0xE6	Loop for speed notch 7
Notch 8 Loop	0xE7	Loop for speed notch 8
Accel 1-2	0xE8	Sound transition from notch 1 to 2
Accel 2-3	0xE9	Sound transition from notch 2 to 3
Accel 3-4	0xEA	Sound transition from notch 3 to 4
Accel 4-5	0xEB	Sound transition from notch 4 to 5
Accel 5-6	0xEC	Sound transition from notch 5 to 6
Accel 6-7	0xED	Sound transition from notch 6 to 7
Accel 7-8	0xEE	Sound transition from notch 7 to 8
Startup	0xEF	Startup sound
Decel 2-1	0xF0	Sound transition from notch 2 to 1
Decel 3-2	0xF1	Sound transition from notch 3 to 2
Decel 4-3	0xF2	Sound transition from notch 4 to 3
Decel 5-4	0xF3	Sound transition from notch 5 to 4
Decel 6-5	0xF4	Sound transition from notch 6 to 5
Decel 7-6	0xF5	Sound transition from notch 7 to 6
Decel 8-7	0xF6	Sound transition from notch 8 to 7
Shutdown	0xF7	Shutdown sound

6.2.18 Optional Triggered Sounds

The automated sound playback SOUNDFX_PLAY_IDX_MOTOR mode has 5x optional event triggered sounds that will play when certain motor speed criteria are satisfied. These are summarized in the table below:

File	Reserved File ID	Description
Change Direction	0xFA	Triggered when motor direction changed
Set Off from Stop	0xFB	Triggered when starting off from stop
Rapid Accel Loop	0xFC	Triggered with rapid acceleration
Rapid Decel Loop	0xFD	Triggered with rapid deceleration
Brake to Stop	0xFE	Triggered with rapid deceleration below a certain speed

6.2.18.1 Change Direction This sound is triggered when the motor direction is changed. A file with file ID 0xFA will automatically playback one time after a change in motor direction.

6.2.18.2 Set Off from Stop This sound is triggered immediately after the motor speed is increased from a stopped (speed=0) state. A file with file ID 0xFB will automatically playback one time after the speed increases from rest in any direction.

6.2.18.3 Rapid Acceleration This sound is triggered when the motor acceleration exceeds a predefined threshold set in the PFx Brick configuration value Rapid Accel Thr (see PFX_CMD_SET_CONFIG). This sound effect may or may not playback depending on how fast the motor speed is increasing. This can be useful for simulating turbo charged prime movers whereby the whining sound of the turbo charger can be played on top of the prime mover sound for enhanced simulation of enhanced load. A file with file ID 0xFC will automatically loop/repeat playback until the motor acceleration reaches zero, i.e. constant speed.

6.2.18.4 Rapid Deceleration This sound is triggered when the motor deceleration exceeds a predefined threshold set in the PFx Brick configuration value Rapid Decel Thr (see PFX_CMD_SET_CONFIG). This sound effect may or may not playback depending on how fast the motor speed is increasing. This can be useful for simulating dynamic brake systems which activate to slow down a locomotive prior to the main braking system. A file with file ID 0xFD will automatically loop/repeat playback until the motor acceleration reaches zero, i.e. constant speed.

6.2.18.5 Brake to Stop This sound is triggered when both the motor deceleration exceeds a threshold (Brake Rate Thr) and the motor speed is below a threshold (Brake Speed Thr) (see PFX_CMD_SET_CONFIG). This sound effect can simulated the sound of brake squeal sounds during the final phase of a locomotive or vehicle stopping. A file with file ID 0xFE will automatically loop/repeat playback until the motor speed reached zero speed/stopped.

6.2.19 SOUND_PARAMx

	7	6	5	4	3	2	1	0
14	SOUND_PARAM1							
15	SOUND_PARAM2							

Some sound f/x actions have associated parameters and are encoded as follows:

ID	MNEMONIC	SOUND_PARAM1	SOUND_PARAM2
0x00	SOUNDFX_NONE		
0x01	SOUNDFX_INC_VOLUME		
0x02	SOUNDFX_DEC_VOLUME		
0x03	SOUNDFX_SET_VOLUME		VOLUME
0x04	SOUNDFX_PLAY_ONCE	RETRIGGER	RELVOLUME
0x05	SOUNDFX_PLAY_CONTINUOUS		RELVOLUME
0x06	SOUNDFX_PLAY_NTIMES	REPEAT_COUNT	RELVOLUME
0x07	SOUNDFX_PLAY_DURATION	DURATION	RELVOLUME
0x08	SOUNDFX_PLAY_PITCHBEND_MOTOR	MOTOR_OUTPUT	GAIN
0x09	SOUNDFX_PLAY_GATED_MOTOR	MOTOR_OUTPUT	GAIN
0x0A	SOUNDFX_PLAY_AM_MOTOR	MOTOR_OUTPUT	GAIN
0x0B	SOUNDFX_STOP		
0x0C	SOUNDFX_PLAY_IDX_MOTOR	MOTOR_OUTPUT	IDX_OPTIONS
0x0D	SOUNDFX_PLAY_RAND	PROBABILITY	
0x0E	SOUNDFX_FILE_SEEK	TIME_MSB	TIME_LSB
0x0F	SOUNDFX_FILE_SCRUB	TIME_MSB	TIME_LSB

6.2.20 SOUND_PARAMx Definitions

6.2.20.1 DURATION The DURATION parameter specifies a fixed time interval used by some f/x.

Value	Definition	Value	Definition
0x0	0.5 sec	0x8	15 sec
0x1	1.0 sec	0x9	20 sec
0x2	1.5 sec	0xA	30 sec
0x3	2.0 sec	0xB	45 sec
0x4	3.0 sec	0xD	60 sec
0x5	4.0 sec	0xD	90 sec
0x6	5.0 sec	0xE	2 min
0x7	10 sec	0xF	5 min

6.2.20.2 RETRIGGER If an event to playback the same file occurs while the file is playing, the RETRIGGER parameter specifies which action should be taken as follows:

```
0 = Toggle playback on/off
1 = Restart playback from the beginning of the file
```

6.2.20.3 REPEAT_COUNT A numeric value specifying the number of times to repeat audio playback. The valid range is 1 to 100.

6.2.20.4 VOLUME A numeric value specifying audio volume. The valid range is 0 to 255 corresponding to minimum and maximum volume respectively.

6.2.20.5 RELVOLUME 2's complement 0x8 ~ 0x7 corresponding to a relative volume level expressed as a gain/attenuation factor in dB from the current playback volume.

6.2.20.6 GAIN The GAIN parameter corresponds to the gain or amount of influence motor speed has on the modulation. The valid range is -100 to 100, where negative values define modulation effect in the opposite sense to motor speed, e.g. audio volume which decreases with increasing motor speed.

6.2.20.7 MOTOR_OUTPUT The MOTOR_OUTPUT parameter specifies which motor channel is used to modulate a sound f/x. Motor channels A, B, C, and D are specified as 0x0, 0x1, 0x2, and 0x3 respectively.

If the MOTOR_OUTPUT parameter is used with the SOUNDFX_PLAY_IDX_MOTOR sound Fx, then bit 2 of MOTOR_OUTPUT can specify whether the desired motor channel's target or current speed is used to determine the index of the sound file to play.

If MOTOR_OUTPUT[2] = 0 then the target speed is used, if MOTOR_OUTPUT[2] = 1 then the current speed is used.

6.2.20.8 IDX_OPTIONS The IDX_OPTIONS parameter customizes the behaviour of the SOUNDFX_PLAY_IDX_MOTOR sound Fx. The IDX_OPTIONS parameter is defined as follows:

Startup Sound Override allows any changes in motor speed to interrupt the playback of startup sounds. This is a useful option to avoid waiting for a lengthy startup sound to finish. If set to 1, motor speed changes will halt playback of the startup sound, and immediately start operational motor sounds. If not set (0), the startup sound file will playback to completion before responding to any motor speed changes for sound playback.

Play Startup Shutdown specifies whether or not sound files representing engine startup and shutdown sounds should be played when the SOUNDFX_PLAY_IDX_MOTOR sound Fx is toggled on or off. If set to 1, sound files representing the startup and shutdown sounds should be pre-loaded into the file system with the correct corresponding reserved file IDs.

Volume Modulation specifies if any volume modulation should also be applied to motor indexed sound playback. This will allow for a variable amount of loudness to be simulated corresponding to engines which sound louder at higher speeds.

```
0x0 = no volume modulation
0x1 = light modulation
0x2 = medium modulation
0x3 = heavy modulation
```

6.2.20.9 PROBABILITY The PROBABILITY parameter specifies the approximate probability of playing a specified sound file when used with the SOUNDFX_PLAY_RAND sound Fx. The PROBABILITY parameter is specified as follows:

```
0x0 = rare
0x1 = occasional
0x2 = often
0x3 = very often
```

6.2.20.10 TIME_MSB TIME_LSB The TIME_MSB and TIME_LSB values represent a 16 bit two's-complement time value in units of 100 ms per LSB. For the EVT_SOUNDFX_FILE_SEEK action this represents an absolute time position in the file. For the EVT_SOUNDFX_FILE_SCRUB action, this represents a relative time offset from the current playback position. Therefore the complete range of values that can be represented are:

```
0x7FFF = 3276.7 sec
0x000A = 1.0 sec
0x0001 = 0.1 sec
0x0000 = 0 sec
0xFFFF = -0.1 sec
0xFFF6 = -1.0 sec
0x8000 = -3276.8 sec
```

7 Notifications

The PFx Brick implements an optional notification mechanism to asynchronously send messages to a connected host. These notification messages operate on a subscription model whereby the host indicates which combination of notifications it wants to receive. After a command has been issued to subscribe to notifications, the PFx Brick will then send messages corresponding to the desired notification events. The notifications can be enabled or disabled at any time by the host.

The specify which notifications are desired to be sent, a logical-OR combination of bit flags is used. This allows for any desired combination of notifications to be sent to the host as required. The flags to specify notifications are defined as follows:

ID	MNEMONIC	Description
0x01	PFX_NOTIFICATION_AUDIO_PLAY_DONE	When any audio channel reaches the end of its playback interval, a notification is sent with a parameter indicating which audio file ID ended playback.
0x02	PFX_NOTIFICATION_AUDIO_PLAY	When an audio channel begins playback, a notification is sent indicating which audio file ID is starting playback.
0x04	PFX_NOTIFICATION_MOTORA_CURR_SPD	Periodic notifcations are sent indicating the current speed of motor channel A
0x08	PFX_NOTIFICATION_MOTORA_STOP	A notification is sent when motor channel A stops
0x10	PFX_NOTIFICATION_MOTORB_CURR_SPD	Periodic notifcations are sent indicating the current speed of motor channel B
0x20	PFX_NOTIFICATION_MOTORB_STOP	A notification is sent when motor channel B stops
0x40	PFX_NOTIFICATION_TO_BLE	Instructs the PFx Brick to send notifications to the Bluetooth LE interface
0x80	PFX_NOTIFICATION_TO_USB	Instructs the PFx Brick to send notifications to the USB interface

For example, if a BLE connected host wants to receive notifications for audio stop events and motor channel A and B speed changes, then the command message would be as follows:

0	1
0x60	0x55

to disable notifications completely, the following command message is used:

0	1
0x60	0x00

8 Memory Map

The PFx Brick has non-volatile flash memory storage used to store its configuration and audio files. Typically, the PFx Brick can come configured with 4, 8, or 16 MBytes of flash storage. This is partitioned into the following regions:

4 MB		8 MB		16 MB	
Address	Memory Space	Address	Memory Space	Address	Memory Space
0x000 000	File system	0x000 000	File system	0x000 000	File system
	...		...		...
0x3FB FFF		0x7FB FFF		0xFFB FFF	
0x3FC 000 0x3FD FFF	FAT Sector Map	0x7FC 000 0x7FD FFF	FAT Sector Map	0xFFC 000 0xFFD FFF	FAT Sector Map
0x3FE 000 0x3FE FFF	FAT Directory	0x7FE 000 0x7FE FFF	FAT Directory	0xFFF 000 0xFFE FFF	FAT Directory
0x3FF 000 0x3FF 1FF	Config space	0x7FF 000 0x7FF 1FF	Config space	0xFFF 000 0xFFF 1FF	Config space
0x3FF 200 0x3FF 9FF	Event LUT	0x7FF 200 0x7FF 9FF	Event LUT	0xFFF 200 0xFFF 9FF	Event LUT
0x3FF A00 0x3FF FFF	Reserved	0x7FF A00 0x7FF FFF	Reserved	0xFFF A00 0xFFF FFF	Reserved

9 Flash Memory File System

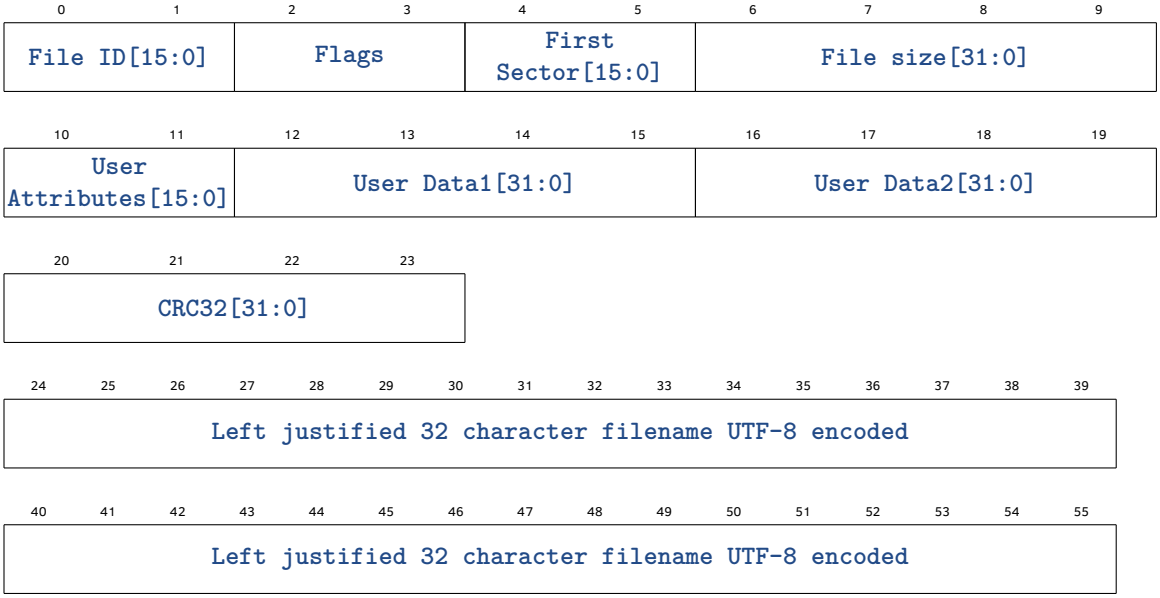
The majority of the capacity of PFx Brick flash memory is dedicated to storing a simple block-oriented file system. This file system allows files of any content to be transfered to and from the connected USB host. The primary function of this file system is to store audio files; however, it is general purpose enough to be used for storage of any file type for future applications.

Access to the file system is provided by a set of conventional file I/O methods such as open, close, read, write, etc. Before any file can be accessed, it must be opened. This will ensure that pointers to the file data content for read and write operations are initialized to a known state. Open files must also be closed when the host has completed any read or write tasks. This ensures any buffered data is safely committed back to the file system and the state of file handles and directories remain consistent.

The details of allocating files across the flash memory is completely abstracted and managed by the file system. The file system automatically allocates space for new files, performs garbage collection on freed/deleted files, pre-erases blocks of flash memory for instant allocation, and arbitrates access to the flash memory from all sources.

9.1 Flash Directory Structure

A file system directory contains a list of the files stored as well as several fields of meta data associated with each file. The format of individual flash directory entries is as follows:



9.1.1 File ID

The `File ID` is a unique identifier which is used to identify and distiguish files. It can have any value in the range 0x0000 to 0x7FFE. An identifier value of 0xFFFF signifies an empty directory entry. **Note:** that all file access commands described in this ICD use the **lower 8-bits of the `File ID` only.** The `File ID` is stored as a 16-bit value; however, access requests are made using the lower 8-bits.

Therefore, `File ID` values should be specified as values between 0x00 and 0xFE. The use of the full 16-bits of `File ID` may be exploited in future applications.

9.1.2 Flags

The `Flags` field is used internally within the file system during file operations and is not normally useful to connected host applications.

9.1.3 First Sector

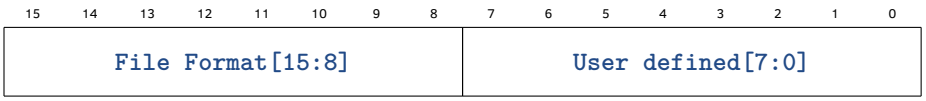
The `First Sector` field points to the location in flash memory of the first sector of the associated file's payload data. This sector location is also used by the file system as a pointer to the beginning of File Allocation Table (FAT) sector chain belonging to the file. Sectors are nominally 4096 byte containers and file data is stored in an integral number of these 4k sectors.

9.1.4 File Size

The `File Size` reports the total number of bytes contained in the file.

9.1.5 User Attributes

The `User Attributes` field stores file specific meta data as follows:



The upper byte stores the `File Format` identifier. Rather than using a typical dotted string extension to the filename, the file format can be optionally stored in the `User Attributes[15:8]` field using a code which maps the file format. The current list of defined file format extensions are as follows:

Value	Definition	Value	Definition
0x00	WAV	0x10	TXT
0x01	FLAC	0x11	HEX
0x02	MP3	0x20	ZIP
0x03	OGG	0x21	GZ
0x04	AU	0x30	PFX
0x05	GSM	0x50	IMG

The user defined bits of `User Attributes` can be used by the host application and firmware in any agreed upon way. Currently, the `User Attributes` field has definitions when associated with WAV files and with text files used for scripting.

WAV File User Attributes

Currently, for the storage of audio WAV files the `User Attributes[7:0]` bits have the following definitions:

7	6	5	4	3	2	1	0
Reserved	Loop Type		Loop Index			Sample Size	Sample Rate

where:

Bit	Definition
0	Sample rate : 0=22.050 kHz, 1=11.025 kHz
1	Sample size : Quantization 0=16 bits per sample, 1=8 bits per sample
4:2	Loop Index when Loop Type[1:0] is not zero
6:5	Loop Type where
	01 = Fixed motor/engine loop sound at power notch
	10 = Accelerating motor sound between power notches
	11 = Decelerating motor sound between power notches

A more detailed description of setting these file attributes for use with motor speed indexed sound effects can be found in Sound F/X Notes in the Action Encoding section.

Script File User Attributes

When a text file is specifically intended to be used for scripting, then an optional value of **0x80** can be assigned to the `User Attributes` field. This is not strictly required for execution of script files; however it can be a useful marker for host applications to easily distinguish ordinary text files from script files.

9.1.6 User Data1/2

The `User Data1` and `User Data2` fields are user defined 32-bit containers for any meta data that either the host or firmware application needs to store conveniently with the file directory entry. These fields are currently defined when used to store audio WAV files as follows:

Field	Definition
`User Data1`	Number of sample bytes in audio WAV file starting at data chunk offset
`User Data2`	Offset in bytes from the start of the file to sample byte data

Note that the PFx Brick firmware automatically fills the contents of `User Attributes`, `User Data1`, and `User Data2` automatically when a WAV audio file is written to the file system.

9.1.7 CRC32

The CRC32 field is a 32 bit hash code automatically generated by the PFx Brick after a file has been written to the file system. This hash code is automatically computed along the entire stream of data bytes of the file. This code can be a useful integrity check of the data that is actually written to the file system. It can also be used loosely as a unique hashing code to verify the identity of a file; however, CRC32 codes are prone to "code collision" for hashing purposes when a large number of files need to be compared.

9.1.8 Filename

The filename field can be used to store a filename containing up to 32 UTF-8 characters. The filename is not used for file directory lookup as with other traditional file systems; rather the File ID field is used for lookup.

9.2 File System Access Commands

The file system is accessed by the host with a group of commands supporting many of the conventional file access tasks. Files are accessed by first opening a handle to a file specified by its unique File ID. When a handle has been obtained, read and write operations may be performed on the file. Finally, after file I/O has been completed, the file handle can be closed. Note that the file handle is not a physical token which is passed to the host, it is effectively a virtual state. When a handle is opened, the PFx file system initializes read and write pointers to a file and applies any subsequent read or write requests to the requested file. It will continue in this state until the handle is closed. The handle is logically associated with the USB interface instance that the host uses to connect to the PFx Brick. There can be up to 4 USB HID interface sessions available and one virtual file handle is associated with each USB HID interface. Connecting with multiple interface sessions allows for a potential increase in transfer bandwidth between the PFx Brick and the host.

The PFX_CMD_FILE_OPEN command opens a virtual file handle to a file for host file I/O. If the specified file does not exist, then it is created by reserving a directory entry for the file and empty storage sectors are allocated for the file. Unlike other file systems, the creation of a new file requires that the file size be known in advance for pre-allocation of sectors in the FAT.

Another consideration when using the file system is that files are currently "Write Once Only". That is, when a file is written, it must not be changed. If changes are required, then the file should be deleted and a new file created to replace it. This differs from other file systems that support arbitrary write access modes to a file. The reason for this restriction relates to the requirement of flash memory to be erased before it can be written. The file system performs routine garbage collection by pre-erasing all memory sectors that have been marked as "free". This lets the file system easily pre-allocate new files immediately for writing. It is possible that the file system may evolve with additional buffering capabilities to support more arbitrary file write schemes; however, this will come at the cost of additional complexity and performance. Nonetheless, despite this restriction, files can be written in any arbitrary sequence of full sectors as long as they are written one time and as one complete sector. Furthermore, write operations should be performed monotonically in increasing byte order. These considerations will likely not be restrictive since files are typically written sequentially from the beginning. Lastly, read operations have absolutely no restrictions in terms of size and sequence. Any number of bytes can be read in random access fashion.

The USB host commands to access the file system are summarized as follows (details for each command can be found in the Host Command Messages reference section):

Command	Definition
PFX_CMD_FILE_OPEN	Open virtual file handle
PFX_CMD_FILE_CLOSE	Close file handle
PFX_CMD_FILE_READ	Read data from file to host
PFX_CMD_FILE_WRITE	Write host data to file
PFX_CMD_FILE_SEEK	Move file pointer to a specified byte offset with respect to the beginning of a file
PFX_CMD_FILE_DIR	Query the file system for directory information or make changes to directory data
PFX_CMD_FILE_REMOVE	Remove a file from the file system
PFX_CMD_FILE_FORMAT_FS	Erase all files and reinitialize the file system directory and file allocation table
PFX_CMD_FILE_GET_FS_STATE	Reports low-level status information on the file system

10 Product ID Codes & Descriptors

Part Number	Product Descriptor	Description
0x1201	PFx Brick alpha	First pre-production prototype PFx Brick with 2x motor channels (using the DRV8839), 8x light channel with discrete pico light connectors, and sound.
0x1202	PFx Brick beta	Second pre-production prototype PFx Brick with 2x motor channels (using the DRV8835), 8x light channels on the standard 10-pin lighting dock connector, and sound.
0x1203	PFx Brick gamma	Third pre-production prototype with 2x motor channels (using the DRV8833), 8x light channels on the standard 10-pin lighting dock connector, and sound.
0x1204	PFx Brick delta IR	Fourth pre-production prototype with 2x motor channels (using the DRV8833), 8x light channels on the standard 10-pin lighting dock connector, and sound.
0x9204	PFx Brick delta	Fourth pre-production prototype with 2x motor channels (using the DRV8833), Bluetooth interface, 8x light channels on the standard 10-pin lighting dock connector, and sound.
0x2204	**PFx Brick IR 4 MB**	Production version of the 4 MB PFx Brick IR with 2x motor channels, 8x light channels, and sound.
0x2208	**PFx Brick IR 8 MB**	8 MB PFx Brick IR
0x2216	**PFx Brick IR 16 MB**	16 MB PFx Brick IR
0xA204	**PFx Brick 4 MB**	Production version of the 4 MB PFx Brick with Bluetooth interface, 2x motor channels, 8x light channels, and sound.
0xA208	**PFx Brick 8 MB**	8 MB PFx Brick
0xA216	**PFx Brick 16 MB**	16 MB PFx Brick
0x1701	PFXLite alpha	Pre-production economy PFx Brick with light f/x only (8x channels with 10-pin dock connector). It has no plastic enclosure, but has stud mounting holes for integration into a model.
0x2702	**PFXLite**	Production economy PFx Brick with light f/x only.
0x1401	PFx Brick Pro alpha	Pre-production PFx Brick with 4x motor channels, 8x light channels, and sound.
0x2404	**PFx Brick Pro 4 MB**	Production 4 MB PFx Brick with 4x motor channels, 8x light channels, and sound.
0x2408	**PFx Brick Pro 8 MB**	8 MB PFx Brick Pro
0x2416	**PFx Brick Pro 16 MB**	16 MB PFx Brick Pro

11 Status Codes

Code	MNEMONIC
0x00	PFX_STATUS_NORMAL
0x33	PFX_STATUS_NORMAL_PENDING
0x55	PFX_STATUS_SERVICE
0x53	PFX_STATUS_SERVICE_PENDING
0x5B	PFX_STATUS_SERVICE_BUSY

12 Error Codes

Several USB command messages include status feedback bytes which may report error or status conditions. Note that there are some error codes which can refer to more than one condition; however, these codes are used in different contexts and therefore will not conflict. For example, some codes reported by the PFX_CMD_GET_STATUS message will be different than the PFX_CMD_FILE_OPEN message. The error codes are summarized as follows:

Code	MNEMONIC
0x00	PFX_ERR_NONE
0x00	PFX_ERR_VERIFY_PASS
0x01	PFX_ERR_VERIFY_FAIL
0x00	PFX_ERR_TRANSFER_REQUEST_OK
0x02	PFX_ERR_TRANSFER_FILE_EXISTS
0x03	PFX_ERR_TRANSFER_TOO_BIG
0x04	PFX_ERR_TRANSFER_INVALID
0x04	PFX_ERR_SPKR_SHORTCIR_FAULT
0x06	PFX_ERR_TRANSFER_CRC_MISMATCH
0x08	PFX_ERR_DAC_OVERTEMP_FAULT
0x0B	PFX_ERR_BLE_FAULT
0x05	PFX_ERR_TRANSFER_FILE_NOT_FOUND
0x06	PFX_ERR_TRANSFER_CRC_MISMATCH
0x07	PFX_ERR_TRANSFER_BUSY_WAIT
0x08	PFX_ERR_TRANSFER_LUT_FULL
0xFF	PFX_ERR_TRANSFER_ERROR
0x80	PFX_ERR_UPGRADE_FAIL
0x0A	PFX_ERR_TRAP_BROWNOUT_RST
0x10	PFX_ERR_TRAP_CONFLICT
0x20	PFX_ERR_TRAP_ILLEGAL_OPCODE
0x40	PFX_ERR_TRAP_CONFIG_MISMATCH

File system access commands have a several error response codes usually passed back as a status byte in a response packet. These error codes are summarized as follows:

Status	Code	Description
0x00	PFX_ERR_NONE	file system operation ok
0xF0	PFX_ERR_FILE_SYSTEM_ERR	overall file system error
0xF1	PFX_ERR_FILE_INVALID	file request was invalid or file is invalid
0xF2	PFX_ERR_FILE_OUT_OF_RANGE	file access request is outside of file size
0xF3	PFX_ERR_FILE_READ_ONLY	file creation or write access denied
0xF4	PFX_ERR_FILE_TOO_BIG	requested file creation is too big
0xF5	PFX_ERR_FILE_NOT_FOUND	requested file ID is not found
0xF6	PFX_ERR_FILE_NOT_UNIQUE	requested file creation ID is already used
0xF7	PFX_ERR_FILE_LOCKED_BUSY	file system is locked or busy
0xF8	PFX_ERR_FILE_SYSTEM_FULL	file system full
0xF9	PFX_ERR_FILE_SYSTEM_TIMEOUT	file access operation time out
0xFA	PFX_ERR_FILE_INVALID_ADDRESS	file system request resulted in an invalid memory address
0xFB	PFX_ERR_FILE_NEXT_SECTOR	file system FAT points to an invalid sector
0xFC	PFX_ERR_FILE_ACCESS_DENIED	file system operation denied or prohibited
0xFF	PFX_ERR_FILE_EOF	file access has reached the end of the file