NZXTSharp Documentation

Release latest

Mar 04, 2019

Welcome to NZXTSharp's readthedocs page!

NZXTSharp is a .NET Standard package that facilitates interaction with NZXT devices. You can find NZXTSharp on Nuget.org, and the source on GitHub.

Docs here are organized by namespace.

NZXTSharp's syntax is designed to be simple and easy to use, almost like python. The structure is built around devices, effects, and params.

Effects are created with params, and effects are applied to devices. Devices are the main point that the user of the SDK will interact with.

A basic getting started example with the Hue+:

```
using NZXTSharp;
using NZXTSharp.Devices;
using NZXTSharp.Effects;

HuePlus hue = new HuePlus();

Fixed effect = new Fixed(new Color(255, 255, 255)); // Create effect object
hue.ApplyEffect(hue.Both, effect); // Apply the effect
```

CHAPTER 1

NZXTSharp

Table of Contents

1.1 Classes

1.1.1 Color.cs

The Color class is used by all NZXTSharp effects, and represents a Color

Constructors

Color()

Color(int R, int G, int B)

Creates a Color instance from the provided R, G, B values.

Values must be 0 - 255 (inclusive).

Color(string hexColor)

Creates a Color instance from the provided hex color code.

Supports color codes with a leading #: `#ffffff`, or without: `ffffff`.

CHAPTER 2

NZXTSharp.COM

The `NZXTSharp.COM` namespace contains classes used for facilitating communication between INZXTDevices and their respective hardware.

Nothing in the COM namespace is meant to be accessed by the user of NZXTSharp. All classes serve infrastructural purposes.

Table Of Contents

2.1 Interfaces

2.1.1 ICOMController.cs

Currently empty.

2.2 Classes

2.2.1 SerialController.cs

Inherets ICOMController.cs

Properties

SerialPort Port { get; } The serial port operated by the SerialController.

SerialCOMData StartData { get; } The SerialCOMData used to start the SerialController.

bool IsOpen { get => Port.IsOpen; } Whether or not the SerialPort operated by the SerialController is open.

Constructors

SerialController(string[] PossiblePorts, SerialCOMData Data)

- param string[] PossiblePorts - A string array containing the ports to attempt opening. Format: “COM3”.
- param SerialCOMData Data - A SerialCOMData object containing the information needed to open the port.

Methods

byte[] Write(byte[] buffer, int responselength) Writes the bytes in the given buffer, and returns the device’s response.

- param byte[] buffer - The bytes to write to the device.
- param int reponselength - The expected length of the response: dictates the size of the returned buffer. Improper sizing will result in lost data.

void WriteNoResponse(byte[] buffer) Writes the bytes in the given buffer to the device. Does not return a response.

- param byte[] buffer - The bytes to write to the device.

void Reconnect() Disposes of and reinitializes the SerialController instance.

void Dispose() Disposes of the SerialController instance.

2.2.2 SerialCOMData.cs

Properties

System.IO.Ports.Parity Parity { get; } The parity type to use.

System.IO.Ports.StopBits StopBits { get; } The stopbits to use.

int WriteTimeout { get; } The write timeout in ms.

int ReadTimeout { get; } The read timeout in ms.

int Baud { get; } The baud rate to open the port with.

int DataBits { get; } The number of databits to use.

Constructors

The SerialCOMData class only has one constructor, and takes arguments corresponding to each of the available properties. The name parameter is optional, and defaults to an empty string.

CHAPTER 3

NZXTSharp.HuePlus

Table of Contents

3.1 HuePlus.cs

Represents an NZXT Hue+ device.

inherits: IHueDevice.cs

3.1.1 Properties

string Name { get; } The device's product name.

Channel Both { get; } A Channel object representing both channels on the Hue+

Channel Channel1 { get; } A channel object representing the Channel 1 of the Hue+ device.

Channel Channel2 { get; } A channel object representing the Channel 2 of the Hue+ device.

List<Channel> Channels { get; } A List containing all Channel objects owned by the Hue+ device.

string CustomName { get; set; } A custom name for the HuePlus device instance.

NZXTDeviceType Type { get; } The NZXTDeviceType of the HuePlus device. `NZXTDeviceType.HuePlus`

3.1.2 Constructors

HuePlus() Constructs a basic HuePlus instance.

HuePlus(int MaxHandshakeRetry = 5, string CustomName = null) Constructs a HuePlus instance with a custom retry count, an

- param int MaxHandshakeRetry - Defaults to 5
- param string CustomName - Deafults to null.

3.1.3 Methods

void Reconnect() Disposes of and reinitializes to the HuePlus instance's COMController.

void Dispose() Disposes of the HuePlus instance's COMController.

void ApplyEffect(Channel channel, IEffect effect, bool SaveToChannel = true) Applies a given IEffect to a given Channel.

- param Channel channel - The HuePlus Channel to apply the affect to. Must be owned by the same HuePlus instance the effect is being applied to.
- param IEffect effect - The IEffect to apply.
- param bool SaveToChannel - Whether or not to save the given IEffect to the given Channel. Defaults to true.

void ApplyCustom(byte[] Buffer) Writes a custom buffer to the HuePlus instance's COMController.

void UnitLedOn() Turns on the Hue+ device's unit led.

void UnitLedOff() Turns off the Hue+ device's unit led.

void SetUnitLed(bool State) Sets the Hue+ device's unit led based on the `State` param.

- param bool State - Which state to set the LED to. true: on, false: off.

void UpdateChannelInfo(Channel Channel) Updates the given Channel's ChannelInfo.

- param Channel Channel - The Channel instance to update.

3.2 HuePlusChannel.cs

Channels are “owned” by devices. Channels also “own” a number of ISubDevices, and a ChannelInfo object.

3.2.1 Properties

int ChannelByte { get; } The Channel instance’s ChannelByte.

IEffect Effect { get: } The IEffect currently applied to the Channel isntance.

bool State { get; } Whether or not the Channel instance is active (on).

ChannelInfo ChannelInfo { get; } The Channel’s ChannelInfo object.

IHueDevice Parent { get; } The device that owns the Channel.

List<ISubDevice> SubDevices { get; } A list of ISubDevices owned by the Channel.

3.2.2 Constructors

Channel(int ChannelByte) Constructs a Channel instance with the given ChannelByte.

- param int ChannelByte - The Channel’s ChannelByte; 0x00 for both, 0x01 for Channel 1, 0x02 for Channel 2.

Channel(int ChannelByte, IHueDevice Parent) Constructs a Channel instance with a given ChannelByte, owned by a given Pare

- param int ChannelByte - The Channel’s ChannelByte; 0x00 for both, 0x01 for Channel 1, 0x02 for Channel 2.
- param IHueDevice Parent - The IHueDevice that will own the Channel object.

Channel(int ChannelByte, IHueDevice Parent, ChannelInfo Info) Constructs a Channel instance with a given ChannelByte, owr

- param int ChannelByte - The Channel’s ChannelByte; 0x00 for both, 0x01 for Channel 1, 0x02 for Channel 2.
- param IHueDevice Parent - The IHueDevice that will own the Channel object.
- param ChannelInfo Info - A ChannelInfo object about the Channel.

3.2.3 Methods

void RefreshSubDevices() Refreshes all ISubDevices in the Channel’s SubDevices list.

void On() Turns the Channel on, and re-applies the last applied effect.

void Off() Turns the Channel off, and applies an “#000000” fixed effect.

void UpdateChannelInfo() Updates the Channel instance’s ChannelInfo object.

void SetChannelInfo(ChannelInfo Info) Sets the Channel instance’s ChannelInfo to a given ChannelInfo “Info”.

3.3 HuePlusChannelInfo.cs

Users of NZXTSharp are generally not meant to construct ChannelInfo instances.

3.3.1 Properites

int NumLeds { get; } The total number of LEDs available on a Channel.

int NumSubDevices { get; } The number of ISubDevices available on a given Channel.

NZXTDeviceType Type { get; } The type of ISubDevices available on a Channel.

3.3.2 Constructors

ChannelInfo(byte[] data) Constructs a ChannelInfo instance from some data returned from a channel handshake.

3.3.3 Methods

void Update() Updates the information contained in a ChannelInfo object.

3.4 CandleLight.cs

Inherits from NZXTSharp.IEffect

EffectName = "CandleLight"

3.4.1 Constructors

CandleLight(Color Color) Constructs a CandleLight effect.

- param Color Color - The Color to display.

3.5 Direction.cs

The direction param is used to specify the direction some effects move in, sometimes defining whether the effect moves smoothly or not.

Inherits from NZXTSharp.IParam

3.5.1 Constructors

Direction(bool isForward, bool withMovement) Constructs a Direction param with the given bool values.

- param bool isForward - Whether or not the effect will move forward or backward.
- param bool withMovement - Whether or not the effect will move smoothly.

CHAPTER 4

NZXTSharp.KrakenX

Table of Contents

4.1 KrakenX.cs

Represents an NZXT KrakenX device.

Inherits: NZXTSharp.INZXTDevice

4.1.1 Properties

Type	Name	Access	Description
HIDDeviceID	DeviceID	{ get; }	The HIDDeviceID of the KrakenX device.
NZXTDeviceType	Type	{ get; }	The NZXTDeviceType of the KrakenX device.
int	ID	{ get; }	A unique device ID.
KrakenXChannel	Both	{ get; }	Represents both the Logo, and Ring channels.
KrakenXChannel	Logo	{ get; }	Represents the KrakenX's logo RGB channel.
KrakenXChannel	Ring	{ get; }	Represents the KrakenX's ring RGB channel.
System.Version	FirmwareVersion	{ get; }	The KrakenX device's firmware version.

4.1.2 Constructors

KrakenX() Constructs a KrakenX instance.

4.1.3 Methods

CHAPTER 5

NZXTSharp.Exceptions

Table of Contents

5.1 Classes

5.1.1 IncompatibleDeviceTypeException

thrown When an NZXTDeviceType is passed to a method or constructor that is not compatible with that method or constructor.

5.1.2 IncompatibleEffectException

thrown When an effect passed to a device's `ApplyEffect()` method is not compatible with that device.

5.1.3 IncompatibleParamException

thrown When a param object passed to an effect constructor is not compatible with that effect.

5.1.4 InvalidEffectSpeedException

thrown When an invalid speed value is passed to a param or effect constructor.

Speed values must be 0 - 4 (inclusive); 0 being slowest, 2 being normal, and 4 being fastest.

5.1.5 MaxHandshakeRetryExceededException

thrown When the maximum number of handshake attempts has been exceeded during device intitialization.

Max Retry Count is `5` by default.

5.1.6 SubDeviceDoesNotExistException

thrown When a user attempts to reference a subdevice that does not exist.

Ex: If there are only four fans connected to a given channel (SubDevices highest index: 3), and the user attempts to reference `Channel.SubDevices[4]`, this exception will be thrown.

5.1.7 SubDeviceLEDDoesNotExistException

thrown When a user attempts to reference a subdevice LED that does not exist.

Ex: If there is only one strip connected to a given channel (SubDevices.Leds highest index: 9), and the user attempts to reference `Channel.SubDevices[0].Leds[10]`, this exception will be thrown.

5.1.8 TooManyColorsProvidedException

thrown When a Color[] of length greater than 8 is passed to an effect constructor.

CHAPTER 6

Getting Started with NZXTSharp

This page contains getting started examples and code snippets.

NZXTSharp's syntax is designed to be simple and easy to use, almost like python. The structure is built around devices, subdevices, effects, and params.

Effects are created with params, and effects are applied to devices. Devices can own subdevices.

An example of a subdevice is a fan, or RGB strip. Once Hue 2 support is added, the CableComb and Underglow SubDevices will be added.

Table of Contents

6.1 Hue+

The Hue+ model consists of the HuePlus class instance. The instance owns three Channel objects, but only Channel1 and Channel2 have SubDevices.

6.1.1 Boilerplate

To get started, you'll need the following:

```
using NZXTSharp.Devices;

HuePlus hue = new HuePlus();
```

With this HuePlus instance, you can apply effects, get channel info, toggle the unit LED, etc.

6.1.2 Applying Effects

Effects are applied to a Hue+ device with the `ApplyEffect()` method. The channel(s) to apply the effect to, and the effect object are passed as params to this method.

Adding on to the last example:

```
using NZXTSharp;
using NZXTSharp.Devices;
using NZXTSharp.Effects;

HuePlus hue = new HuePlus();

HexColor color = new Color(255, 255, 255); // You can make colors with RGB values
HexColor color = new Color("#ffffff"); // Also works with Hex codes (with or without␣
↪the leading #)

Fixed effect = new Fixed(color); // Create effect

hue.ApplyEffect(hue.Both, effect); // Apply effect to both channels
hue.ApplyEffect(hue.Channel1, effect); // Or just one

hue.UnitLedOff(); // Turn unit LED off
hue.UnitLedOn(); // And back on again!
```

6.1.3 SubDevices

A HuePlus instance has two Channels that own subdevices: Channel1 and Channel2. The Both Channel does not own any because it is not a "physical" channel. Keep in mind that ALL changes to subdevices need to be "pushed" to the device by setting, or re-setting an effect.

All subdevice classes are held in the `NZXTSharp.Devices` namespace.

Building on the Boilerplate example:

```
using NZXTSharp.Devices;
using NZXTSharp.Effects;

HuePlus hue = new HuePlus();
Fixed effect = new Fixed(new Color("#FFFFFF"));

List<ISubDevice> Ch1Devices = hue.Channel1.SubDevices; // Not necessary, but syntax␣
↪looks better

Ch1Devices[0].AllLedOff(); // Turn off all LEDs on first subdevice in channel.
hue.ApplyEffect(hue.Channel1, effect); // Apply changes
```

(continues on next page)

(continued from previous page)

```
Ch1Devices[1].ToggleLed(9); // Toggle individual LEDs with the ToggleLed method
Ch1Devices[1].Leds[8] = false; // Or by setting the value.
hue.ApplyEffect(hue.Channel1, effect); // Apply changes

Ch1Devices[1].ToggleLedRange(1, 5); // Or, toggle ranges of LEDs
hue.ApplyEffect(hue.Channel1, effect); // Apply changes
```

6.1.4 Custom RGB values for a Fixed Effect

The fixed effect, being the most versatile, allows the user of NZXTSharp to construct a fixed effect with custom RGB values for each LED. This is done by passing a byte array to a Fixed effect constructor.

Byte array schema: The byte array must have at least 1 RGB value, and at most 120 RGB values. If the length is not within this range, an InvalidParamException will be thrown.

RGB Value formatting: For the effect to display properly, all RGB values MUST be passed in G, R, B format. RGB Values, like all other RGB values in NZXTSharp must be between 0-255 (inclusive).

Building on the Boilerplate example:

```
using NZXTSharp;
using NZXTSharp.Devices;
using NZXTSharp.Effects;

HuePlus hue = new HuePlus();

// Create RGB array, this will have two LEDs lit: one white, one red.
byte[] colors = new byte[] { 255, 255, 255, 0, 255, 0 };
Fixed effect = new Fixed(colors); // Create Effect

hue.ApplyEffect(hue.Both, effect); // And apply it!

// Also supports subdevice and LED toggling.
hue.Channel1.SubDevices[1].ToggleState(); // Toggle device
hue.ApplyEffect(hue.Both, effect); // Apply changes
```

CHAPTER 7

Hue+ Protocol

Table of Contents

Basic Protocol Schema: Command Type, ChannelByte, EffectByte, Param1, Param2, LedData ChannelByte: Both = 00, Channel 1 = 01, Channel 2 = 02

Command Types: Set Effect = 4b, Unit LED = 46, Get Channel Info = 8d

The Hue+ operates on a serial port, and is made to handle discrete commands sent in packets. To open a connection to a Hue+ device, open a serial connection on whatever COM port your Hue+ is operating on with a baud rate of `256000`, parity set to `None`, dataBits set to `8`, and stopBits set to `1`. Then, begin the handshake process.

Effect protocols are made of exactly 125 bytes or less. For all protocols, the first five bits in each packet are what I will call "settings bytes", and the remaining 120 are LED data in G, R, B format.

Settings bytes (in order) consist of which kind of command is being set, the channels to apply the effect to, which effect to set, and two parameters. See set effect protocol for more information.

7.1 Handshakes

To begin interaction with a Hue+ device, a handshake must first be completed.

The "Hello" handshake can be completed by continuously sending 0xc0, until the Hue+ unit reponds with 0x01.

There is no trick to a "GoodBye" handshake, just close the serial connection.

7.2 Channel Handshakes/ Getting Channel Info

To get information about what is connected to a channel, send an `8d ChannelByte` command to the Hue+. Ex: To get channel info for channel 1, send `8d 01`. For channel 2, `8d 02`. The response structure is still being worked out, some of the values are still unclear.

The response should be 5 bytes long, and follows this schema:

- ? : ? Consistent between devices
- ? : ? Not consistent between devices
- ? : ? Not consistent between devices.
- X : Fan or Strip; 0x00 = strips, 0x01 = fans.
- X : Number of fans or strips connected.

7.3 Set Effect

Below is a table outlining the settings packets for each effect. Bolded param values are defined below in the Param Scemas Section.

Direction params marked with *WM* can make use of movement in the effect. See the direction param schema below for more information.

Effect	Packets/ Send			EffectByte	Param1	Param2
Fixed	1	0x4b	CB	0x00	0x03	0x02
Fading	1/ Color*	0x4b	CB	0x01	0x03	**CIS/S**
Spectrum Wave	1	0x4b	CB	0x02	**Direction**	**CIS/S**
Marquee	3	0x4b	CB	0x03	**Direction**	**LS/S**
Covering Marquee	3/ Color*	0x4b	CB	0x04	**Direction**	**CIS/S**
Alternating	2	0x4b	CB	0x05	**Direction WM**	**CIS/S**
Pulse	1/ Color*	0x4b	CB	0x06	0x03	**CIS/S**
Breathing	1/ Color*	0x4b	CB	0x07	0x03	**CIS/S**
Candle Light	1	0x4b	CB	0x09	0x03	0x02
Wings	1	0x4b	CB	0x0c	0x03	**CIS/S**

7.4 Param Schemas

7.4.1 CIS/S - Color In Set/ Speed

CIS/S params are a composite of a couple values: The index of the current color in a set of colors, and the speed of the effect. Find the values individually, and concatenate them to get the btye to be passed as a param.

- First Digit: Color In Set. If there are multiple colors being applied, this digit denotes the index of the color.
  - To Find: digit = $x * 2$
    - x: The color number (Zero Indexed)
- Second Digit: Speed
  - 0 - 4 where 0 is slowest, and 4 is fastest. 2 is normal.
- IF Effect only uses one color, first digit is 0.
- Whole Byte: Concatenate Color In Set (IN HEX), and Speed.
  - Ex: If the effect uses one color, and was at normal speed, the resulting byte would be *02*.
  - Ex: If the color is the third one in the set, and the speed is at fastest, the resulting byte would be *44*.

7.4.2 Direction

For direction, just like CIS/S, the byte result is a composite of two values: whether or not the effect's direction is forward or backward, and whether or not the effect should be moving.

If an effect's param1 byte is marked with *WM*, it can make use of movement toggling.

The byte values are as follows:

- Forward: 03
- Backward: 13
- IF marked as *WM*, the following are also available:
  - Forward &nbsp;&nbsp;W/ Movement: 0b
  - Backward W/ Movement: 1b

7.4.3 LS/S - LED Size/ Speed

To find the desired byte composite, use the table below:

Speed v ; LED Size >	3	4	5	6
Slowest	00	08	10	18
Slow	01	09	11	19
Normal	02	0a	12	1a
Fast	03	0b	13	1b
Fastest	04	0c	14	1c

7.5 Unit LED Protocols

Turning the Hue+ unit's LED on or off is pretty simple. All of the data needed fits into one packet, and seven bytes.

Just send the desired byte codes over the serial port, and the light should do as instructed.

On: 46 00 c0 00 00 00 ff

Off: 46 00 c0 00 00 ff 00

Special Thanks to Pet0203. **for helping me get started and providing base code.**

CHAPTER 8

KrakenX Protocol

I want to give a special thanks to Jonas Malaco for his help with building out KrakenX support in NZXTSharp, and his work in reverse engineering the KrakenX protocol.

Table of Contents

This document describes the HID communication protocol for NZXT KrakenX (x42, x52, x62, x72) devices. The vendor ID for NZXT is 0x1e71, and the device id for KrakenX devices is 0x170e.

CAM receives device information about once/ second, I call these "status reports". CAM also consistently sends packets to the Kraken device. The packets sent from CAM are meant to set the pump/ fan speeds based on whatever pump/ fan profile is currently set. Jonas Malaco

8.1 Handshakes

KrakenX devices have no hello or goodbye handshake process.

8.2 Status Reports

The Kraken device continuously sends status reports upstream to the PC. These reports are always 0x40 bytes long. So far, information about how to get pump/ fan speeds, liquid temp, and firmware version. Here is how to get that information from a status report: In the following examples, `data` refers to the array of bytes last received from the device (zero-indexed).

Pump Speed: data[4] << 8 | data[5] - `<<` is the bitwise left-shift operator, and `|` is the bitwise OR operator.

Fan Speed: data[4] * 0x100 + data[5]

Liquid Temp: data[0] + (data[1] * 0.1) - The liquid temp value in degrees C, unrounded.

Firmware Version:

- Major: data[10]
- Minor: (int)data[12].ToString() + data[13].ToString()

8.3 Overrides

If you want custom pump/ fan speeds to be set, the KrakenX device requires "overrides" at least once every 10 seconds, or the device will revert back to the CAM default "performance profile". The override buffer schema is as follows:

Pump: 0x02, 0x4d, 0x40, 0x00, (byte)Speed - Speed is the desired speed as a percentage.

Fan: 0x02, 0x4d, 0x00, 0x00, (byte)Speed - Speed is the desired speed as a percentage.

8.4 Set Effect

The process of setting an RGB effect is similar to how it is with the Hue+. RGB Effect packets are always 65 bytes long. There are 5 settings bytes at the beginning, then 9 G, R, B color codes, then padding out to the 65 length with 0x00. The 9 GRB color codes are for the 8 LEDs in the ring, and the one in the logo. Even when effects are set on just the logo or ring, there are still 9 color codes. The first 8 seem to be for the ring, and the last is for the logo.

The settings bytes schema is as follows: 0x02, 0x4c, Param1, EffectByte, Param2

Below is a table outlining the settings packets for each effect. Bolded param values are defined below in the Param Schemas Section.

Effect Name	Packets/ Send			CB/ Param1	Effect-Byte	Param2	Compatible With
Fixed	1	0x02	0x4c	CB	0x00	0x02	Both
Fading	1/ Color	0x02	0x4c	CB	0x01	**CISS**	Both
SpectrumWave	1	0x02	0x4c	**DCB**	0x02	Speed	Both
Marquee	1	0x02	0x4c	CB	0x03	**LSS**	Ring Only
CoveringMarquee	1/ Color	0x02	0x4c	**DCB**	0x04	**CISS**	Both (at same time), or Ring Only
Alternating	1/ Color	0x02	0x4c	**DCBWM**	0x05	**CISS**	Ring Only
Breathing	1/ Color	0x02	0x4c	CB	0x06	**CISS**	Both
Pulse	1/ Color	0x02	0x4c	CB	0x07	**CISS**	Both
TaiChi	2	0x02	0x4c	CB	0x08	**CISS**	Ring Only
WaterColor	1	0x02	0x4c	CB	0x09	Speed	Ring Only
Loading	1	0x02	0x4c	CB	0x0a	Speed	Ring Only
Wings	1	0x02	0x4c	CB	0x0c	Speed	Ring Only

Speed refers to the speed the effect will display at: 0 - 4 where 0 is slowest, and 4 is fastest. 2 is normal.

8.5 Param Schemas

The KrakenX shares the CISS and LSS param with the Hue+. The KrakenX does have a couple of its own unique (for now) params:

8.5.1 DCB - Direction/ ChannelByte

This param is a concatenation of two ints:

- First Byte: Direction – Forward: 0, Backward: 1
- Second Byte: ChannelByte – The ChannelByte of the channel the effect will be applied to.

8.5.2 DCBWM - Direction/ ChannelByte (With Movement)

This param is only used for the Alternating RGB effect. The desired value can be found with this table:

Direction v; With Movement >	True	False
Forward	0x0A	0x02
Backward	0x1A	0x12

8.5.3 CIS/S - Color In Set/ Speed

CIS/S params are a composite of a couple values: The index of the current color in a set of colors, and the speed of the effect. Find the values individually, and concatenate them to get the btye to be passed as a param.

- First Digit: Color In Set. If there are multiple colors being applied, this digit denotes the index of the color.
  - To Find: digit = $x * 2$
    - x: The color number (Zero Indexed)
- Second Digit: Speed

- 0 - 4 where 0 is slowest, and 4 is fastest. 2 is normal.

- IF Effect only uses one color, first digit is 0.
- Whole Byte: Concatenate Color In Set (IN HEX), and Speed.
  - Ex: If the effect uses one color, and was at normal speed, the resulting byte would be *02*.
  - Ex: If the color is the third one in the set, and the speed is at fastest, the resulting byte would be *44*.

8.5.4 LS/S - LED Size/ Speed

To find the desired byte composite, use the table below:

Speed v ; LED Size >	3	4	5	6
Slowest	00	08	10	18
Slow	01	09	11	19
Normal	02	0a	12	1a
Fast	03	0b	13	1b
Fastest	04	0c	14	1c

CHAPTER 9

Support

To get help with NZXTSharp, you can:

- Email me: akmadian@gmail.com
- Create an issue or look into the source code: https://github.com/akmadian/NZXTSharp
- Join the NZXTSharp Discord server: https://discord.gg/yK8m2CU

If you have suggestions for the documentation or the package itself, please let me know using one of the methods above!

Special Thanks To:

Jonas Malaco (github) For his help with the KrakenX protocol

Pet0203 (github) For his help with development, and the Hue+ protocol.

DarkMio (github) For developing Octopede, which contributed to KrakenX support.

CHAPTER 10

Related Projects

Table of Contents

NZXTSharp is for the NZXT community. I hope that it will be used not just by devs, but that those devs will use NZXTSharp to build applications that many people (even with no dev experience) can use. Because of this, I believe that any and all awareness of NZXT SDKs will be beneficial to the community. Listing of an SDK is not an endorsement.

If I missed your SDK, please contact me! My contact info can be found on the support page.

Here are some other open source SDKs or applications:

10.1 Python

kusti8 - hue-plus; A Windows and Linux driver in Python for the NZXT Hue+.

jonasmalacofilho (NZXTSharp Contributor!) - liquidctl; Cross-platform tool and drivers for liquid coolers and other devices.

akej74 - grid-control; Grid Control is a free and open source alternative to NZXT CAM.

10.2 C#

piet-v - HueLightPlus; Ambilight-clone for NZXT Hue+ LED-Controller

DarkMio - Octopede; A replacement for NZXT Kraken Controller

wongfei - nzxt-grid-driver; Simple Grid+ controller/configurator to get rid of ugly CAM software.

10.3 C++

RBlafford - levd; Daemon with configurable properties that will control NZXT Kraken x61 cooler.

rupor-github - squid; Squid service for NZXT Kraken

VIGGEEN - vlm-krakenx-daemon; Daemon for controlling NZXT Kraken series under Linux distributions.

10.4 C

jaksi - leviathan; Linux kernel module to control and monitor NZXT liquid coolers.

10.5 JavaScript

yetzt - node-krkn; nzxt kraken x52/x62 usb interface controller library for nodejs

MarnixBouhuis - NZXT.js; Control NZXT CAM devices with NodeJS

Sparta142 - OpenCAM; A lightweight alternative to NZXT's CAM software for Kraken devices

10.6 Shell

CapitalF - gridfan; A controller script for the NZXT Grid+ v2 fan controller.

10.7 Java

RoelGo - CamSucks; An alternative control software for the NZXT GRID+

Tankernn - JavaGridControl; Open-Source control software for the NZXT Grid+ and Grid+ v2

CHAPTER 11

Supported Devices and Features

This list is the most current resource for checking which devices and features are and are not supported. If a device is not listed, it is not supported.

The Hue+ is supported, KrakenX products are partially supported, the Hue 2 is most likely next. If you would like to request a specific device or feature, please submit an issue at NZXTSharp's GitHub repo, and tag it with `device-request` or `feature-request`.

The supported features tables use the following schema for column values:

- NI - Not Implemented.
- WIP - Work In Progress/ In Process of Implementation.
- ITT - Implemented, To Test.
- TNW - Tested, Not Working.
- TFW - Tested, Fully Working.

Table of Contents

11.1 Hue+

11.1.1 Device Features

Feature	NI	WIP	ITT	TNW	TFW
Unit LED On					X
Unit LED Off					X
Channel On					X
Channel Off					X
Channel Handshakes					X
Subdevices					X
Subdevice LEDs					X

11.1.2 Effects

Feature	NI	WIP	ITT	TNW	TFW
Fixed					X
Fading					X
Spectrum Wave					X
Marquee					X
Covering Marquee					X
Alternating					X
Pulse					X
Breathing					X
Candle Light					X
Wings					X

11.2 KrakenX

11.2.1 Device Features

Feature	NI	WIP	ITT	TNW	TFW
Get Pump Speed					X
Set Pump Speed					X
Channel On					X
Channel Off					X
Get Fan Speed					X
Set Fan Speed					X
Get Liquid Temp					X
Get Firmware Version					X

11.2.2 Effects

Feature	NI	WIP	ITT	TNW	TFW
Fixed					X
Fading		X			
Spectrum Wave					X
Marquee					X
Covering Marquee		X			
Alternating		X			
Pulse		X			
Breathing		X			
TaiChi					X
Wings					X
WaterCooler					X
Loading					X