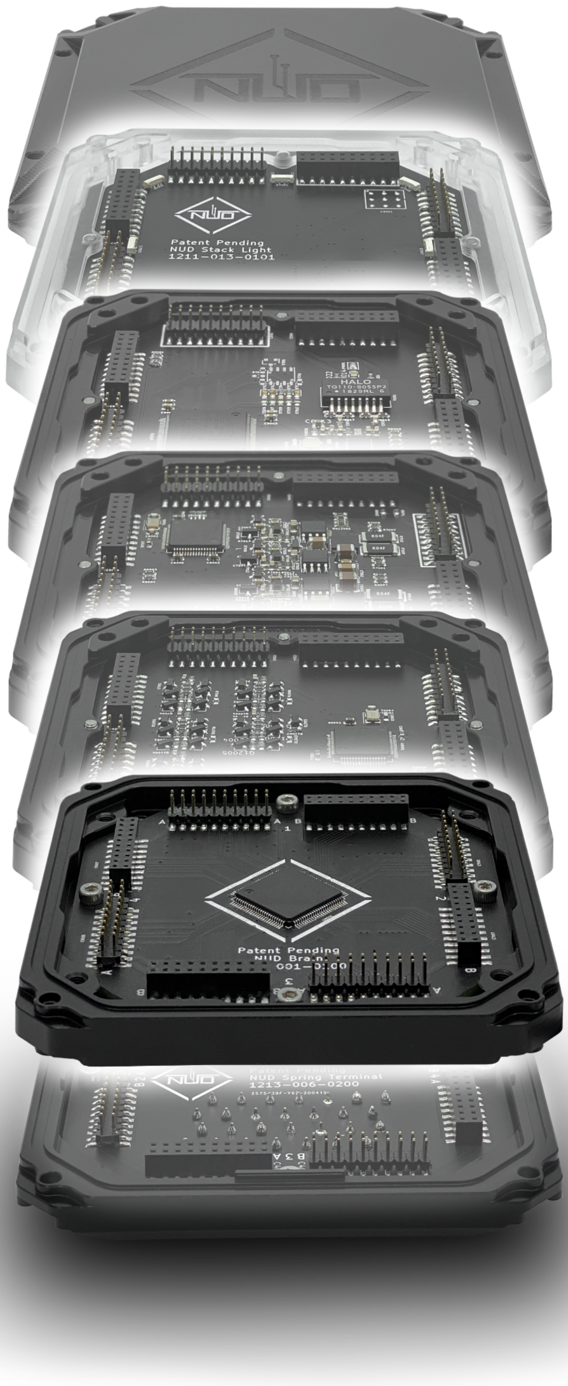
LED Stack
User Manual



https://www.nodeudesign.com
sales@nodeudesign.com

Patent # 10993323

Table of Contents

1.1. Electrical Characteristics

Absolute Maximum Ratings	
	Max
VIN	36V
V(ESD) Electrostatic discharge	16000V
Temperature (Storage & Operating)	-40℃ - 80℃
Operating Temperature	-20℃ - 80℃

Recommended Maximum Ratings			
	Min	**Typical**	**Max**
Operating Voltage	7.5V	-	36V

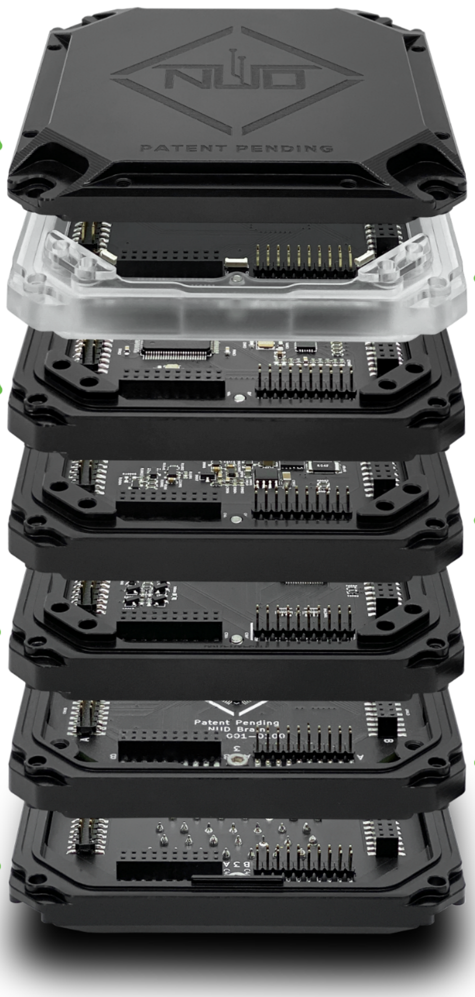
1.2. Stack Configuration

Stack Cover
- IP-67 Rated Dust & Water Resistant
- NUD Flange Set Mountable
- Machined & Anodized Aluminum

Ethernet Peripheral
- TCP, UDP, DHCP Client, Web Server, & MQTT Protocols Supported
- 10 / 100Mbps & Auto MDI-X

Input Peripheral
- 5 Input Channels w/ DSP & Filtering
- Digital, Voltage, Current, PWM, & Encoder Input Modes Available

Spring Terminal Peripheral
- High Quality Terminal Blocks
- M16 & 1/2" Conduit Stack Covers
- I/O for Any Stack Configuration



LED Peripheral
- Sixteen 2800mcd RGB LEDs
- Configurable LED Control Modes
- Configurable Event Triggers

Output Peripheral
- Four Power Outputs & One Aux. Input
- Digital, Voltage, Current & PWM Control Modes with Tuneable PIDs

Brain
- Core Controller of the NUD System
- 216MHz ARM Cortex M7 Processor
- < 10ms Peripheral Response Time

1.3. Spring Terminal Pinout

	Side 4	
🟥	V_{IN}	Power
🟥	V_{IN}	Power
⬛	GND	Ground
⬛	GND	Ground
🟩	A1	Unused
🟩	A2	Unused
🟩	A3	Unused
🟩	A4	Unused
🟩	A5	Unused
🟦	B1	Unused
🟦	B2	Unused
🟦	B3	Unused
🟦	B4	Unused
🟦	B5	Unused

	Side 3	
🟥	V_{IN}	Power
🟥	V_{IN}	Power
⬛	GND	Ground
⬛	GND	Ground
🟩	A1	Input Channel 1
🟩	A2	Input Channel 2
🟩	A3	Input Channel 3
🟩	A4	Input Channel 4
🟩	A5	Input Channel 5
🟦	B1	Unused
🟦	B2	Unused
🟦	B3	Unused
🟦	B4	Unused
🟦	B5	Unused

	Side 2	
🟥	V_{IN}	Power
🟥	V_{IN}	Power
⬛	GND	Ground
⬛	GND	Ground
🟩	A1	Output Channel 1
🟩	A2	Output Channel 2
🟩	A3	Output Channel 3
🟩	A4	Output Channel 4
🟩	A5	Unused
🟦	B1	Unused
🟦	B2	Unused
🟦	B3	Unused
🟦	B4	Unused
🟦	B5	Unused

	Side 1	
🟥	V_{IN}	Power
🟥	V_{IN}	Power
⬛	GND	Ground
⬛	GND	Ground
🟩	A1	Twisted Pair 1 Solid
🟩	A2	Twisted Pair 1 Striped
🟩	A3	Twisted Pair 2 Solid
🟩	A4	Twisted Pair 2 Striped
🟩	A5	Unused
🟦	B1	Unused
🟦	B2	Unused
🟦	B3	Unused
🟦	B4	Unused
🟦	B5	Unused

2. Overview

The LED Stack Module is an all-in-one control system solution designed to give the user a variety of input, output, and status indication abilities fully controllable through Ethernet. A preset UDP packet command structure allows the user to read/write configuration settings and execute operations remotely. An M16 Terminal Cover is included for easy installation on a threaded pipe or corresponding cable gland.

2.1. Hardware

LED Peripheral:

The LED Peripheral is designed to display status or event information about your system. Sixteen 2800 candella RGB LEDs provide a high degree of illumination for the user to maximize noticeability. Multiple LED configuration commands are available to the user - allowing LED color control, blink, and spin modes.
See "*LED Peripheral User Manual*" for full LED Peripheral specifications and information.

Ethernet Peripheral:

The Ethernet Peripheral is the primary medium for which the user communicates and controls the LED Stack. The Ethernet Peripheral supports auto MDI-X, simplifying cabling procedures through automatic transmission negotiation. The Ethernet Peripheral supports up to 10/100 speeds. See "*Ethernet Peripheral User Manual*" for full Ethernet Peripheral specifications and information.

Output Peripheral:

The Output Peripheral consists of four output channels and one auxiliary input channel. The Output Peripheral allows the user to drive resistive or inductive loads, while the auxiliary input channel may be used to measure feedback externally. PWM, Voltage, Current, and Digital drive modes are available for the user to configure. See "*Output Peripheral User Manual*" for full Output Peripheral specifications and information.

Input Peripheral:

The Input Peripheral's five input channels may be configured to capture a wide range of signals, including: Voltage (0-10V), Current(0-20mA), Digital, PWM Frequency and Duty Cycle, and Encoder signals. Not only may the Input Peripheral be used to monitor states of a system, it may be used to drive Events. Input Trigger Events allow the user to configure setpoints for which the Input Peripheral will alert the user if a threshold has been crossed. See *2.2.2 Input Trigger Events*. Also see "*Input Peripheral User Manual*" for full Input Peripheral specifications and information.

Brain:

The Brain is the core of the stack. It runs a Python3 interpreter that processes human readable python files off of the integrated uSD card. Python allows for easy code customization to suit the needs of every application. See "*Brain User Manual*" for full Brain specifications and information. Also see "*Brain Getting Started*" for more information about modifying and writing custom software solutions.

Spring Terminal Peripheral / M16 Cover

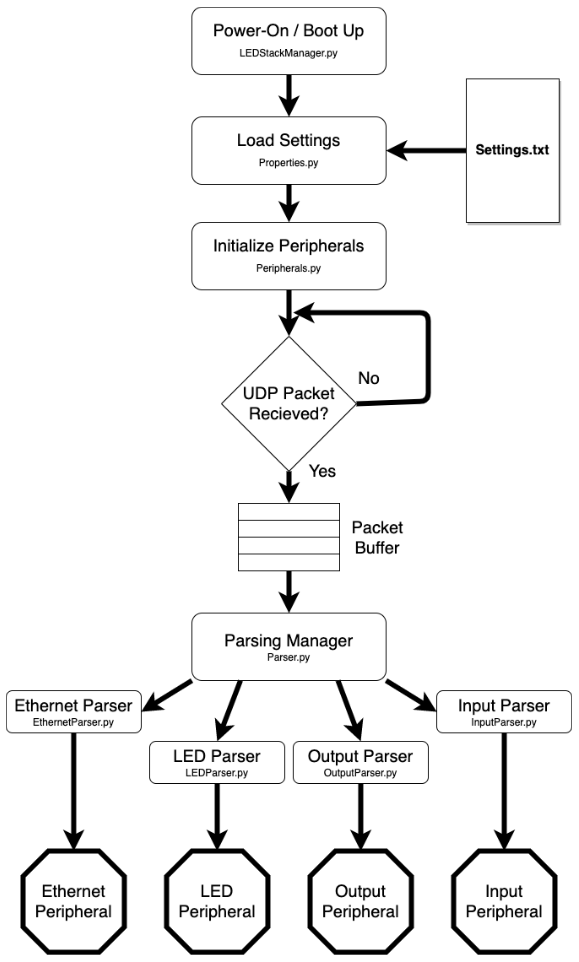
The spring terminal peripheral allows simple wire connections to whatever cable is available. Passing the cable through the cable gland yields a water tight seal for harsh environments. Mounting the LED Stack Module on a threaded pipe allows for highly visible mounting solutions.. See "*Spring Terminal User Manual*" for full Spring Terminal Peripheral specifications and information.

2.2. Software

This section describes the protocol used to read & configure the LED Stack's peripherals, the flow of data to their associated properties through the LED Stack's software, and the methods of which properties are parsed, loaded and saved. The LED Stack's software package is designed to send, receive, and parse UDP packets through Ethernet. When the Ethernet Peripheral has initialized it's Ethernet Server, the LED Stack will remain ready to receive and relay packets to the Brain for processing. All software is stored on the onboard uSD card and may be modified for whatever application is necessary. The factory default software package is available for download on NodeUDesign.com.

2.2.1. UDP Packet Reception

When the LED Stack receives a packet, the packet will be stored in a temporary buffer where it is queued for processing. The packet buffer ensures that packets will be handled as soon as possible, that reception will not interfere with processing of the most recent packet, and that packets will be processed in order of which they are received. Each packet is directed through a multi-staged parser where it is determined if the packet is a valid. The packet's data contents are also examined against the minimum and maximum parameters of the targeted property. If the packet and data check out as valid, the parser will extract the packet's command bytes, read / write byte, and (if applicable) property data. This information will then be directed to the associated property. If the packet is deemed invalid, the packet will be discarded. In either case, once a packet has been processed and the property function call has been completed, processing of the next packet will then begin. For a detailed table of writable properties, see *2.2.5 UDP Packet Structure*.



2.2.2. UDP Packet Transmission

The LED Stack is fully capable of reading / fetching properties from its peripherals and sending back the data to the user. The LED Stack may be instructed to return data by sending a UDP packet containing the desired property command bytes, and setting the Read / Write byte to Read (‘R’, 0x52). When the LED Stack receives the read instruction, it will either pull the data directly from *LEDStack/properties.py*, or fetch the information directly from the desired peripheral. Read response times from peripherals are often less than 10 milliseconds. For a detailed table of readable properties, see *2.2.5 UDP Packet Structure*.

2.2.3. Loading and Saving Settings

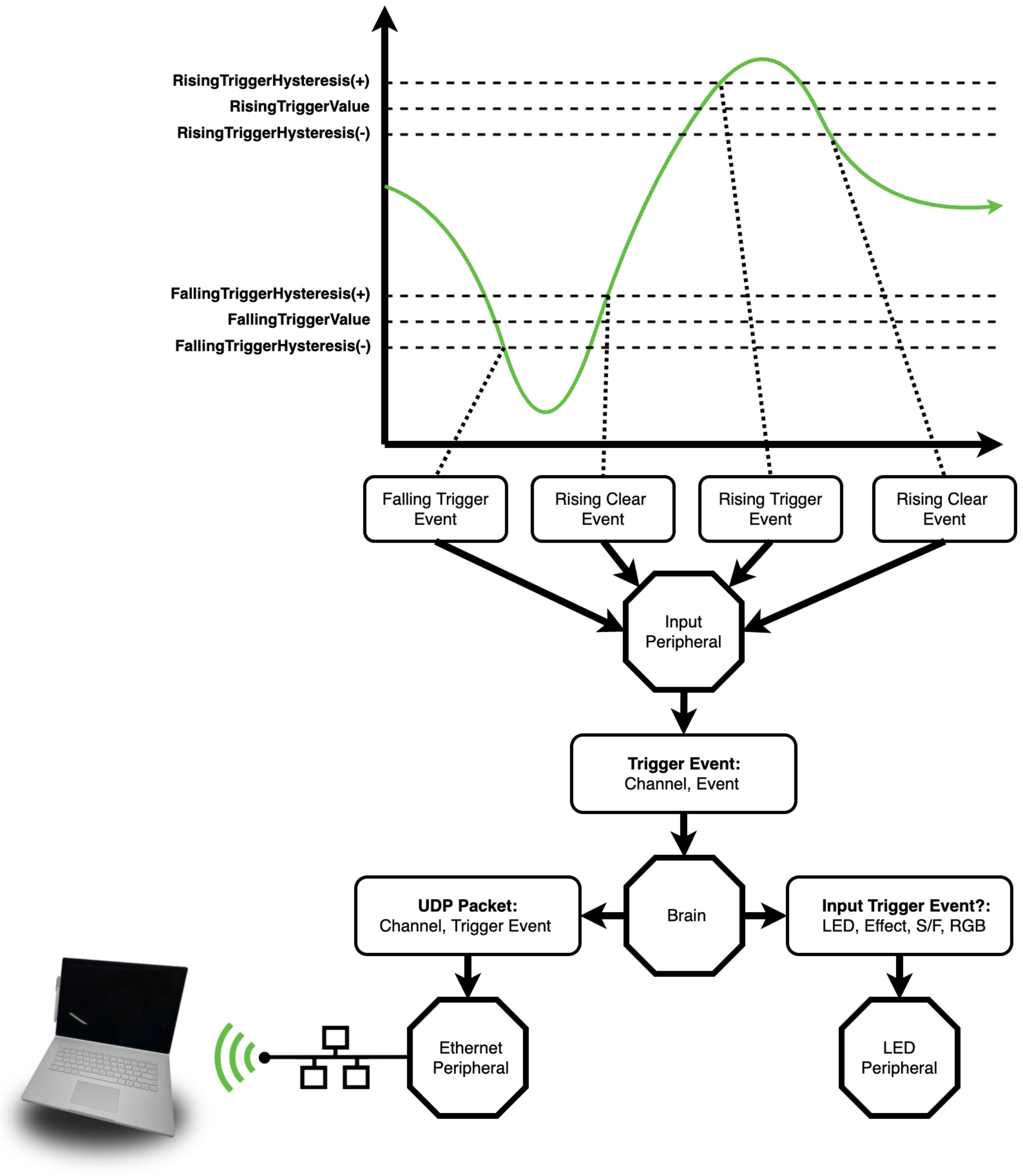
The text file *LEDStack/settings.txt,* located on the Brains uSD card, may be used to pre-load settings at startup, or save and load settings at any time. Upon power-up or a reboot, the LED Stack first parses *LEDStack/settings.txt* and loads its values into *LEDStack/properties.py*. Next, the LED Stack initializes all attached peripherals - Ethernet, LED, Output, and Input Peripherals. When initialization is complete, the LED Stack’s Ethernet interface is brought online. Ethernet Server initialization may take up to ten seconds from startup. The LED Stack will then read *LEDStack/property.py* and configure the stack's peripherals to the user's presets. Finally, the LED Stack will begin listening for UDP packets.

The settings in the settings file follow a few simple formatting rules. Comments are denoted by a *‘#’* as the first character of a line. Multi-line comments are NOT supported at this time. Any empty line will be ignored and may be used for formatting and separating sections. The setting must be given as <*Name*>=<*value*>. White space before, after, and between the text and equal sign will be ignored.

2.2.4. Events

Input Trigger Events:

The Input Peripheral may be configured to continually monitor the states of its inputs and trigger Events when a measured value crosses user-defined threshold values. Input Trigger Events automatically alert the user of the Input Peripheral's channel status. When an Input Trigger Event occurs, the brain creates a UDP packet containing the specific trigger event and the channel information. The packet is then sent to the stored destination IP Address and Port as soon as possible. Input Trigger Events may also be used to automatically modify the state of the LED Stack's LED Peripheral. When enabled, Input Trigger Events may be used in applications where visual status indication can identify the state of a system. See the flowchart below depicting how an event flows through the LED Stack when an event is triggered.


RisingTriggerHysteresis(+)
RisingTriggerValue
RisingTriggerHysteresis(-)
FallingTriggerHysteresis(+)
FallingTriggerValue
FallingTriggerHysteresis(-)
Falling Trigger Event
Rising Clear Event
Rising Trigger Event
Rising Clear Event
Input Peripheral
Trigger Event: Channel, Event
UDP Packet: Channel, Trigger Event
Brain
Input Trigger Event?: LED, Effect, S/F, RGB
Ethernet Peripheral
LED Peripheral

2.2.5. UDP Packet Data Structure

Description:

A properly formatted UDP packet data frame consists of two or three components depending on if the user is reading or writing to a property. If the user wishes to read a property, the generic data packet structure contains two parts: a peripheral property being accessed, and the read byte ('R', 0x52). If the user wishes to write a new value to a property, the generic packet structure contains three parts: the property being accessed, the write byte ('W', 0x57), and the byte array containing the desired properties' new data value.

LED Stack System Functions:

Description	**Property**	**Read/Write**
Reset	0x001	Read Only
	UINT_16	*UINT_8*
Save Settings	0x002	Read Only
	UINT_16	*UINT_8*
Load Settings	0x003	Read Only
	UINT_16	*UINT_8*

Ethernet Peripheral:

Description	**Property**	**Read/Write**	**Data[0]**	**Data[1]**	**Data[2]**	**Data[3]**
Source IP Address	0x101	R/W	ADDR 3	ADDR 2	ADDR 1	ADDR 0
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*
Source Port	0x102	R/W	Port H	Port L		
	UINT_16	*UINT_8*	*UINT_16*			
Destination IP Address	0x103	R/W	ADDR 3	ADDR 2	ADDR 1	ADDR 0
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*
Destination Port	0x104	R/W	Port H	Port L		
	UINT_16	*UINT_8*	*UINT_16*			
MAC Address	0x105	Read Only	ADDR 3	ADDR 2	ADDR 1	ADDR 0
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*
Gateway	0x106	R/W	ADDR 3	ADDR 2	ADDR 1	ADDR 0
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*
NetMask	0x107	R/W	ADDR 3	ADDR 2	ADDR 1	ADDR 0
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*

LED Peripheral:

Description	Property	Read/Write	Data[0]	Data[1]	Data[2]
Effect	0x200	R/W	Effect: 0 = Off 1 = On 2 = Blink 3 = Spin	Effect Speed 0 = Slow 1 = Fast	
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	
Color	0x210 - 0x212	R/W	R	G	B
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*

Output Peripheral:

Description	Property	Read/Write	Data[0]
Output Mode	0x301 - 0x304	R/W	Mode: 0 = Inactive 2 = PWM 4 = Current 1 = Digital 3 = Voltage
	UINT_16	*UINT_8*	*UINT_8*
Output Value	0x311 - 0x314	R/W	Dependant on Mode
	UINT_16	*UINT_8*	*UINT_16*
Output Voltage	0x321 - 0x324	Read Only	Output Voltage
	UINT_16	*UINT_8*	*UINT_16*
Output Current	0x331 - 0x334	Read Only	Output Current
	UINT_16	*UINT_8*	*UINT_16*

Input Peripheral:

Description	Property	Read/Write	Data
Input Mode	0x401 - 0x405	R/W	Mode: 0 = Inactive 2 = PWM 4 = Current 1 = Digital 3 = Voltage
	UINT_16	*UINT_8*	*UINT_16*
Input Value	0x411 - 0x415	Read Only	Dependant on Mode
	UINT_16	*UINT_8*	*UINT_16*
Input Rising Trigger Value	0x421 - 0x425	R/W	Dependant on Mode
	UINT_16	*UINT_8*	*UINT_16*
Input Rising Trigger Hysteresis	0x431 - 0x435	R/W	Dependant on Mode
	UINT_16	*UINT_8*	*UINT_16*
Input Rising Trigger	0x441 - 0x445	EVENT	
	UINT_16		
Input Rising Trigger Clear	0x451 - 0x455	EVENT	
	UINT_16		
Input Falling Trigger Value	0x461 - 0x465	R/W	Dependant on Mode
	UINT_16	*UINT_8*	*UINT_16*
Input Falling Trigger Hysteresis	0x471 - 0x475	R/W	Dependant on Mode
	UINT_16	*UINT_8*	*UINT_16*
Input Falling Trigger	0x481 - 0x485	EVENT	
	UINT_16		
Input Rising Clear	0x491 - 0x495	EVENT	
	UINT_16		

Input Trigger Event	0x4A1 - 0x4A5	R/W	Rising / Falling Edge	LED Number: 1-4	LED Effect: 0-3	LED Effect Speed: 0 = Slow 1 = Fast	R	G	B
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*
Input Clear Event	0x4B1 - 0x4B5	R/W	Rising / Falling Edge	LED Number: 1-4	LED Effect: 0-3	LED Effect Speed: 0 = Slow 1 = Fast	R	G	B
	UINT_16	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*	*UINT_8*

1.1. Creating a Python 3 UDP Packet Sender

```
# Python 3

import socket
import struct

TestPacket = struct.pack(‘!HBBBB’, 0x211, 0x57, 0, 255, 0)
ServerAddress = (“192.168.1.15”, 2000)
LEDStackAddress = (“192.168.1.20”, 2000)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(ServerAddress)

s.sendto(TestPacket, LEDStackAddress)

# In order to open a socket and send UDP packets to the LED Stack using a Python 3 IDE,
the user must first import two libraries: socket and struct. We create three objects
containing a test packet (in this case the LEDColor packet), the server address (your
computer, or sending device), and the LEDStacks configured IP Address (sourceIPAddress in
LEDStack/settings.txt. Next, the code creates a socket for which to send UDP Packets
over, and binds the device's address to it. The device then sends the test packet to the
LED Stacks IP Address.
```

1.2. Configuring the Output Peripheral’s Channel 2 to PWM with 50% Duty Cycle

Method 1: Parsing Settings.txt

```
# from: settings.txt
# outputMode = 0, 2, 0, 0
# outputDesiredValue = 0, 500, 0, 0

# In Python 3, sending device

LoadSettingsPacket = struct.pack('!HB', 0x001, 0x57)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(LoadSettingsPacket, LEDStackAddress)

# The above code first looks at LEDStack/settings.txt, where the user has defined
outputModes channel 2 as '2' (PWM Mode), and outputDesiredValue as '500' (50% Duty Cycle)
The user saves the file and opens a Python 3 IDE. The code then creates a structured
packet, LoadSettingsPacket, and passes in the proper command byte and write byte (0x001,
0x57). The code then creates a socket for which UDP packets may be sent, then sends the
packet.
```

Method 2: Configuring the Output Peripheral via Ethernet UDP

```
# Python 3

OutputCh2ModeWrite = struct.pack(‘!HBB’, 0x302, 0x57, 2)
OutputCh2DesiredValue = struct.pack(‘!HBH’, 0x312, 0x57, 500)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(OutputCh2ModeWrite, LEDStackAddress)
s.sendto(OutputCh2DesiredValue, LEDStackAddress)

# The above code first creates objects containing packet structures. Note the “!”,
declaring the byte format of the struct as big-endian. The code then creates the socket
for which UDP packets may be sent. The code then sends the two packets to the LED Stack,
where they will be processed. The Output Peripheral’s Channel 2 should now be configured
in PWM Mode with 50% Duty Cycle.
```

1.3. Reading the Input Peripheral’s Channel 1 Mode and Value

```
Python 3

InputCh1ModeRead = struct.pack(‘!HB’, 0x401, 0x52)
InputCh1ValueRead = struct.pack(‘!HB’, 0x411, 0x52)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(InputCh1ModeRead, destIPAddress)
s.sendto(InputCh1ValueRead, destIPAddress)
```

```
Output: 2
Output: 770

# The above code first creates objects containing packet structures to read the Input
Peripherals Channel 1 Input Mode and InputValue. The code then creates the socket for
which UDP packets may be sent. The code then sends the two packets to the LED Stack,
where they will be processed. The LED Stack fetches the values from the Input Peripheral,
and returns a UDP Packet containing the requested properties.
```