Wireless Addressable Strings of Pixels (WASP) User Guide

John van Schouwen

v1.0, 2019

Table of Contents

1 Introduction

1.1 Overview

A Wireless Addressable Strings of Pixels (WASP) network provides a seasonal lighting technology that uses power efficient, low voltage (12 Vdc) strings of individually addressable tricolour LEDs, or RGB pixel strings. Pixel strings, pictured in *Figure 1*, are now readily available in retail stores and in online Web sites such as Ebay and Amazon; they come in WS2811 or WS2812 varieties[1]. The pixel strings are controlled directly by WASP Lighting Slave units that are, in turn controlled and coordinated by a central WASP Lighting Controller unit. The controller is user-programmable so that a set of fixed and animated lighting effects sequences, suitable for any holiday or for year-round ambiance lighting, can be composed.

Figure 1: RGB Pixel String

WASP slaves are powered by a 12 Vdc power source that drives both the slave unit and its connected pixel string(s). The required power source capacity is determined by the number of LED pixels that it needs to drive. The WASP controller is powered by a USB (5 Vdc) power source and can be connected to, say, a computer or a USB wall charger. The controller has a long range (> 150 metres) providing great flexibility in its location relative to its slave units.

1 WS2811 and WS2812 refer to the type of chips used internally on the pixel strip for driving the LEDs.

1.2 Features

The following are the WASP technology features:

- 433 MHz radio with range > 150 metres
- Support for up to 5 slaves per controller
- Power sources:
  - WASP Lighting Controller unit: 5 Vdc via micro USB connector
  - WASP Lighting Slave unit: 12 Vdc
    - Power source capacity needs are determined by the connected pixel string(s).
- Lower power consumption:
  - WASP Lighting Controller unit: 40 mA, 200 mW @ 5 Vdc
  - WASP Lighting Slave unit (without pixel string): 42 mA, 500 mW @ 12 Vdc
  - Pixel string:
    - Per tricolour LED: 7 mA, 80 mW @ 12 Vdc
    - 5 m pixel string of 150 LEDs: 1.05 A, 13 W @ 12 Vdc
- Remote wireless WASP slave configuration:
  - Slaves can be configured via the WASP controller connected via USB serial port to any computer.
- Remote wireless firmware upgrades on WASP slaves:
  - Slaves can be upgraded using a Moteino Wireless Programming unit
- Programmable WASP Lighting Controller unit:
  - WASP Interactive Programming Environment (WIPE)
  - Interpreted WIPE programming language
  - 4 KiB of onboard EEPROM for storing multiple programs
  - Automatically run a selected program upon power-up
- LED indicators:
  - WASP Lighting Controller unit:
    - Blinking red power indicator
    - Green indicator while WIPE is running
    - Blinking blue indicator while running a WIPE program
  - WASP Lighting Slave unit:
    - Blinking red power indicator
    - Blinking green indicator while receiving WASP commands
    - Blinking blue indicator while transmitting WASP commands
- Freely available, open source software and development environment
  - Firmware for controller, slaves, and wireless programmer use open source Arduino code

- Controller, slaves, and wireless programmer use Moteino hardware
  - Moteino is an Arduino compatible microcontroller board that can include radios (with various operating frequencies and ranges) and onboard flash memory for wireless firmware upgrades.
  - The current release of WASP uses RFM69W (433 MHz) radios:
    - WASP Controller: MoteinoMEGA-USB with radio
    - WASP Slave: Moteino with radio and onboard Flash memory (no USB)
      - Moteino can support up to 100 pixels
      - A MoteinoMEGA could be used to support a greater number of pixels
    - Moteino Wireless Programmer: Moteino-USB with radio
  - Assembled Moteino boards can be purchased at http://www.lowpowerlab.com/shop
    - Current pricing (in US dollars) as of April 2019:
      - Moteino with radio and Flash: $22.90
      - MoteinoMEGA with radio and Flash: $29.95
      - Moteino-USB with radio: $26.90
      - MoteinoMEGA-USB with radio: $29.95
  - Construction of WASP controller and slave units will require some rudimentary knowledge of electronics and soldering skills.

1.3 WASP Network Overview

Figure 1 Illustrates the connectivity in a WASP network. The WASP controller communicates wirelessly with the WASP slave units. Under normal (lighting effects) operation, the controller only

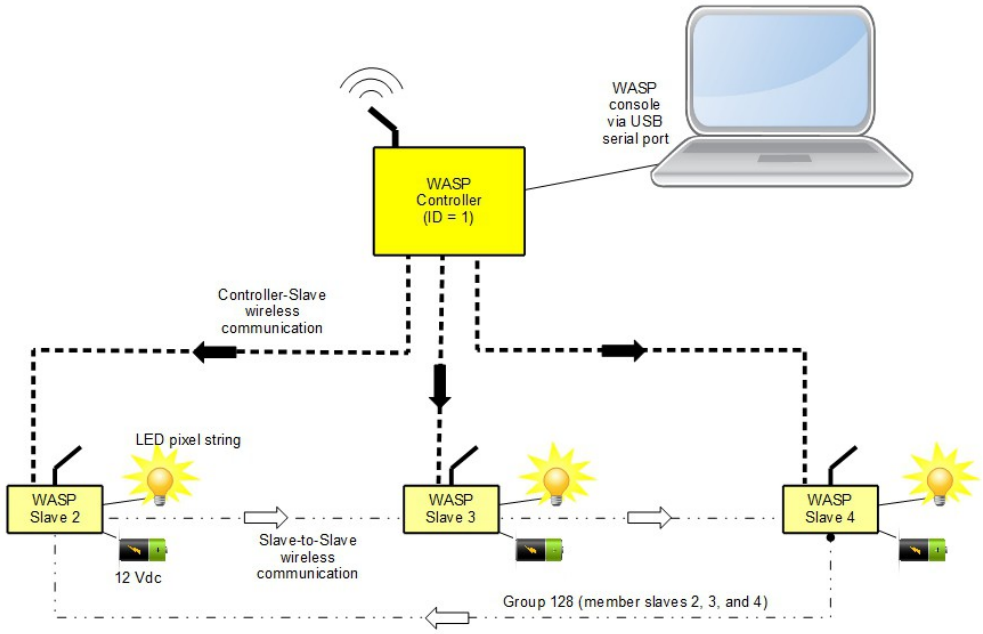


Figure 2: WASP Network Connectivity

transmits commands to the slaves. The slaves can be grouped together into multiple groups, each being a ring consisting of from one to five slaves. In response to a command to shift pixel colours left or right (refer to section *4.3.2.3 The WIPE Functions*), a slave transmits pixel colour data to one neighbor and receives pixel data from another. During network configuration (refer to section *3 WASP Network Setup, Configuration, and Administration*), the WASP controller transmits to the slaves commands to which the slaves respond back.

The controller can operate as a standalone unit when a lighting effects program is configured to run automatically; refer to section *4.1 Automatically Running a Lighting Effect*. The WASP controller needs to be connected to a computer via a USB serial port in order to configure the network and program lighting effects.

1.4 WASP Controller Layout

Figure 3 shows the top of a WASP Lighting Controller unit with the radio antenna extending outwards. There is a single RBG LED providing power and WASP network activity indicators. The red LED blinks every 5 seconds while the unit is powered, but not in WIPE mode. The green LED lights while the unit is in WIPE mode, but not running a WIPE program. The blue LED toggles for every instruction executed while running a WIPE program.

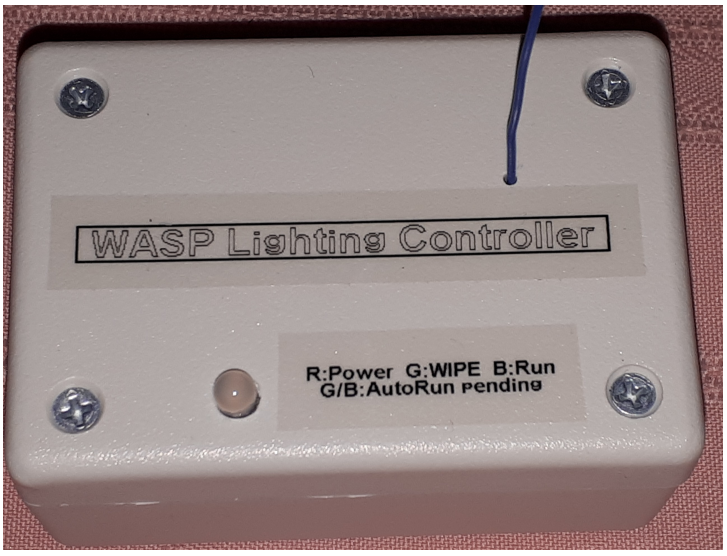


Figure 3: WASP Lighting Controller - Top View

Figure 4 Shows the micro USB port used for powering the controller or for connecting to a computer.

Figure 4: WASP Lighting Controller - Side View

1.5 ***WASP Slave Layout***

Figure 5 shows the top view of a WASP Lighting Slave unit, with the radio antenna extending outwards and a tricolour indicator LED. Extending outward on the right side is the 12 Vdc power input connection cable. And extending outward on the left side is the cable for connecting to a pixel strip as well as a 12 Vdc output that could be used to power a neighboring slave unit. Each cable incorporates an IP65 rated waterproof connector. (Refer to section *2 Connecting a WASP Slave Unit* for information about how to wire up a slave unit.) The LED blinks red every 5 seconds while the unit is powered. The LED toggles in green whenever it receives a WASP command and in blue whenever it transmits a WASP command.

Figure 5: WASP Lighting Slave - Top View

2 Connecting a WASP Slave Unit

This section explains how to wire up a WASP Lighting Slave unit to both its power source, its connected pixel strip(s) and (optionally) neighboring slave units.

A WASP Lighting Slave unit needs to be powered by a 12 Vdc power source that is capable of supplying enough current to power both the slave unit itself and the pixel strip to which it is connected. Each slave is capable of passing its input power thru to the output wires on its control cable. If those connections are used to power additional slave units, be sure that the power source is capable of supplying enough current for all connected slaves and their pixel strips.

Find an appropriate location for mounting the unit. Though the unit is water resistant, it is recommended to mount it away from exposure to the elements to ensure longer life. The connector cables can be extended if needed. However, it is best to keep cable lengths as short as possible and to use heavy gauge wires in order to reduce resistance and the resulting voltage drop. This is especially important for the control connection since the pixel strip data line runs at 5 Vdc and the voltage is impacted more greatly by cable length. Before making any mounting and wiring decisions permanent be sure to run a quick test on each slave node and pixel strip. Use the program in section *4.3.1 First Program: WASP Hello World* to run a test program from the WASP Lighting Control unit.

2.1 Power Side Wiring

Figure 6 shows the power input source connection cable. This is a two-conductor cable, with a black wire for ground (negative) and the red wire for +12 Vdc. *Figure 5* showed the waterproof connector that is incorporated into the cable. The connector can be unscrewed and pulled apart, if necessary, before wiring the free end to the power source.

WARNING

Take appropriate precautions wherever cables, wires, and connections are exposed to the elements (especially water).

Figure 6: WASP Lighting Slave - Right View (power input)

2.2 Control Side Wiring

Figure 7 shows the control and power output connection cable and the cable's wires are shown in *Figure 8*:

- **Blue** and **black** wires are both ground wires (negative)
  - One wire is to be connected to the ground wire (usually white) on the pixel strip
  - The other wire can be used for connecting to the ground (black) wire on a neighboring slave unit.
- **Red** and **yellow** wires are both +12 Vdc
  - One wire is to be connected to the positive supply wire (usually red) on the pixel strip
  - The other wire can be used for connecting to the positive supply (red) wire on a neighboring slave unit.
- The green wire is to be connected to the data wire (usually green) on the pixel strip.

Figure 5 showed the waterproof connector that is incorporated into the cable. The connector can be unscrewed and pulled apart, if necessary, before wiring the free end to the power source. If you are connecting multiple pixel strings to the same slave unit, run separate ground and +12 Vdc cables (using a suitable wire gauge) to each string[2]. Each pixel string must be no more than 5 metres long. At that length, there is too much voltage drop across pixel string's 12 V power lines—to the far end of the string—to be able to adequately power the next pixel string. The data connection must be wired directly from one string to the next with minimal cable length between to two ends.

WARNING

Take appropriate precautions wherever cables, wires, and connections are exposed to the elements (especially water).

Figure 7: WASP Lighting Slave - Left View (control)

2 Those WASP slave units that incorporate a Moteino (rather than a MoteinoMEGA) support only up to 100 pixels. The 12 Vdc pixel strings group together three tricolour LEDs per pixel. Thus a 5 metre pixel string, having 150 LEDs, has actually only 50 pixels.

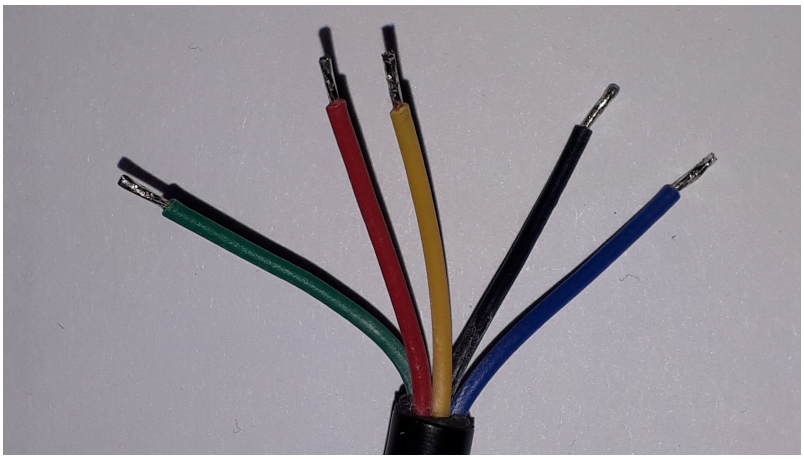


Figure 8: Control and Power Output Wires

***WARNINGS**:*

Be sure to check the pixel strip manufacturer's wiring information. Confirm the colours used for the ground, +12 Vdc, and data wires.

Do not use 5 Vdc pixel strips; they will be damaged when connected to a 12 Vdc power source.

For any unused power output wires be sure to clip off the conductor and cover each with either individual marettes or electrical tape. Shorting these wire could cause a fire and/or damage the 12 Vdc power source.

3 WASP Network Setup, Configuration, and Administration

3.1 Setting up the WASP Network Using the WASP Controller Unit

Once the WASP Lighting Slave units and pixel strings have been wired, tentatively, as per section *2 Connecting a WASP Slave Unit* the slave units need to be configured. The following parameters need to be set:

- Node ID: Each slave must have a unique ID in the range 2 – 6.
- Pixel strip control pin: Unless you have built your own slave, this should be 16.
- Pixel strip parameters:
  - Number of pixels in the connected string(s). The 12 Vdc pixel strings group together three tricolour LEDs per pixel. If the string density is 30 LEDs/metre, then a 5 metre pixel string contains 150 LEDs, but only 50 pixels.
  - The operating frequency: This is either 400 kHz or 800 kHz, depending on the strip.
  - The colour ordering on the pixel strip: Different pixel strips wire the individual colours (red, green, and blue) of each tricolor LED in differing order depending on the manufacturer. This setting allows the slave to adjust for the variations. I use **BRG** order.

To configure, or otherwise administer, the WASP slaves you need to power up the slaves and connect the WASP Lighting Controller unit to the USB port of a computer. Before doing so, follow the instructions in section *3.2 Software Installation* to install the requisite software. Then proceed to section *3.1.1 Connecting the WASP Controller to a Computer* followed by section *3.1.2 Configuring the WASP Slave Units*.

3.1.1 Connecting the WASP Controller to a Computer

The following instructions assume that an FTDI serial port driver and PuTTY have been installed on the computer. (Refer to section *3.2 Software Installation*.)

(1) Connect the micro USB port of the WASP Lighting Controller unit to the USB port of the computer.

(2) Start up PuTTY.

(3) If this is the first time you've connected with PuTTY, continue to the next step. Otherwise, from the “Saved Sessions” list, double click on the session name you had previously saved; you can skip the remaining steps.

(4) In the PuTTY Configuration window, under “Specify the destination you want to connect to”:

- Click the “Serial” radio button.
- Enter 9600 (baud) for the “Speed”.
- Enter the serial port in the “Serial line”
  - On a PC these are named COM1, COM2, etc. If you don't know which is the correct

COM port, open the Device Manager and expand the “Port (COM & LPT)” item. If there is only one item, you're done. Otherwise, while monitoring the list of ports, disconnect and reconnect the controller to see which port dropped out and subsequently returned.

- On Linux there are usually named /tty/USB0, /tty/USB1, etc.

(5) From the left category list in the PuTTY Configuration window, select “Serial” and enter the following data:

- “Serial line to connect to” and “Speed (baud)” should already be set to the serial port and speed you entered in the previous step.
- Set “Data bits” to 8.
- Set “Stop bits” to 1.
- Set “Parity” to “None”
- Set “Flow control” to “None”.

(6) From the left category list in the PuTTY Configuration window, select “Session”. In the “Saved Sessions” field, enter a name that is meaningful to you (e.g. “WASP Controller”). Click the “Save” button to save these settings for future use.

(7) Finally, click the “Open” button to open a terminal session to the WASP Lighting Controller unit.

Once connected, the WASP controller's command console window will pop up and you should see a screen similar to that shown in *Figure 9*.

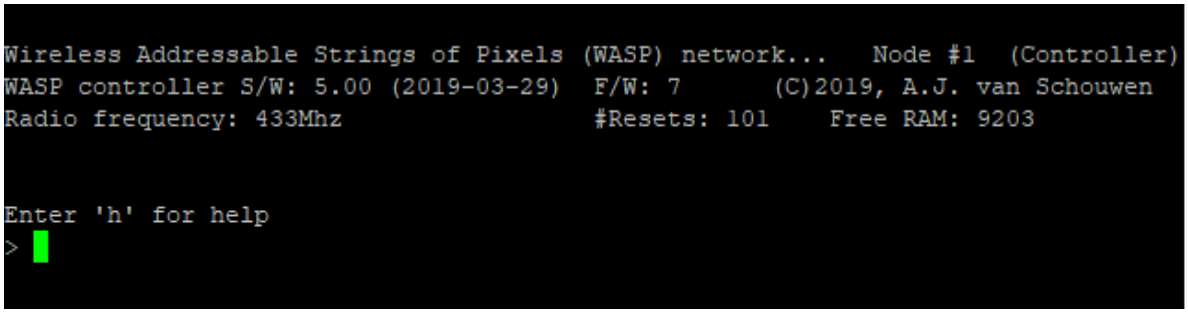


Figure 9: WASP Controller Startup Screen

3.1.2 Configuring the WASP Slave Units

To configure the WASP Lighting Slave units, ensure the slaves are powered up, connect the controller to a PC (refer to section *3.1.1 Connecting the WASP Controller to a Computer*), and perform the following steps:

Preliminary

If, upon power up, the controller's indicator LED is alternating quickly between blue and green, it is configured to automatically run a WIPE program. The controller must be brought into command console mode:

i. If the console displays the `>` prompt, enter the following command in the controller's command console to stop the program from running:

```
autoRun 0
```

ii. Otherwise, the controller will be running the program within WASP Interactive Programming Environment (WIPE) mode. Enter the following sequence of commands:

`x`	(Stops the program, if one is running)
`exit`	(Exits WIPE and returns to the command console)

PART 1: Assign unique node ID values to each slave

The controller's indicator LED should be blinking red about every 5 seconds, confirming that the controller is in command console mode.

(1) Enter the following command to query the current configuration parameters of all slaves:

```
netPing 255
```

If it appears that not all slaves replied, repeat the command a couple of times; then continue to the next step.

(2) If all slaves are listed in the command results, continue on to the next step. Otherwise:

i. Confirm that power is applied to each slave. Each slave's indicator LED should be blinking red every 5 seconds. If not, fix this problem and return to the previous step.

ii. If all slaves are powered, there is likely a node ID conflict where two or more slaves share the same ID. In this case, power down all but one of the slaves. You will need to repeat all of these steps with one slave at-a-time powered up.

(3) Each slave requires a unique node ID in the range 2 - 6[3]. Determine how you wish to number each. Enter the following command to renumber a slave that is currently assigned node ID ***x*** to a new node ID of ***y***:

```
netNodeId x, y
```

Then power cycle that node, or power it off if there were node ID conflicts and you still need to renumber more slaves.

3 The WASP controller always uses a node ID value of 1.

PART 2: Configure each slave

All WASP Lighting Slave units should now be powered up.

(1) Enter the following command to query the current configuration parameters of all slaves:

```
netPing 255
```

The parameter value 255 queries all nodes. You can replace it with a specific slave's node ID to query just that slave.

(2) The pixel strips are normally controlled via pin #16 on the WASP controller's embedded microcontroller. If the pin numbers are already correct on each node, continue on to the next step. Otherwise, to correct the pin number on a specific node (node ID ***x***), enter the following sequence of commands:

```
netLedCtrl x, 16
netSave x              (Saves the configuration setting to the slave's EEPROM)
netReset x
```

If you have built your own WASP slave and it uses a different pin number, replace the value 16 above with the correct value.

(3) Configure each slave's pixel strip parameters. For each node you will need to determine the number of pixels connected to the slave (***n***), the strip's operating frequency (***f***), and the order in which the three colours are wired for the pixels (***c***). Don't worry too much about the colour order for now; you can always return to this step later to correct it once you've determined that there's a mistake. For each slave, (node ID ***x***), enter the following sequence of commands:

```
netLed x, n, f, c
netSave x
netReset x
```

(4) Finally, confirm that all of the parameters are correct by resetting all of the slaves and querying them again:

```
netReset 255
netPing 255
```

And verify that the results are correct.

3.2 Software Installation

To be able to connect the WASP Lighting Controller unit to the computer, you will need have some additional software installed.

Connect the controller's USB port to the computer's USB port. If the driver does not install automatically, you may need to manually install an FTDI USB-to-serial port driver (freely available on the Web).

Accessing the WASP controller requires the use of terminal emulator software. If a terminal emulator (such as TeraTerm or minicom) is already installed, you may use these. However, PuTTY is the recommended emulator and is freely available on the Web.

4 WASP Controller Operation

This section describes the WASP Lighting Controller unit's operation.

The WASP Lighting Controller unit operates in one of two modes—namely, command console mode and WIPE mode. Upon power up, the controller starts off in command console mode.

4.1 Automatically Running a Lighting Effect

When there is at least one lighting effects program stored in the WASP Lighting Controller's EEPROM memory (refer to section *4.3 Programming Lighting Effects on the Controller*), the controller can be configured to automatically run one of the programs after power up. The program to be run is configured in the WASP Interactive Programming Environment (refer to section *4.3.2 The WASP Interactive Programming Environment (WIPE)*). Control of the autorun feature is, however, performed in the command console. To change the autorun settings, perform the following steps:

(1) If, upon power up, the controller's indicator LED is alternating quickly between blue and green, it is configured to automatically run a WIPE program. The controller must be brought into command console mode:

i. If the console displays the `>` prompt, enter the following command in the controller's command console to stop the program from running:

```
autoRun 0
```

ii. Otherwise, the controller will be running the program within WASP Interactive Programming Environment (WIPE) mode. Enter the following sequence of commands:

`x`	(Stops the program, if one is running)
exit	(Exits WIPE and returns to the command console)

The controller's indicator LED should be blinking red about every 5 seconds, confirming that the controller is in command console mode.

(2) Configure the autorun setting:

i. To enable autorun with a default 10 second delay following power up, enter the following command:

```
autoRun 1
```

ii. To enable autorun with a delay, ***x***, in the range of 0 – 255 seconds enter the following command:

```
autoRun 1, x
```

iii. To disable autorun, enter the following command:

```
autoRun 0
```

(3) Save the autorun setting to EEPROM memory by entering the following command:

```
save
```

4.2 Command Console Commands

This section describes the commands that are available while the WASP Lighting Controller is in console command mode.

The command prompt **>** indicates that the controller is in command console mode. (The command prompt ***** indicates WIPE mode; refer to section *4.3.2 The WASP Interactive Programming Environment (WIPE)*.)

While in console command mode, at any time you can get a summary of commands by entering the following command:

h

The command output is shown in *Figure 10*. The commands are as follows:

```
Enter 'h' for help
> h
[h]


Console commands: (spaces can replace commas)
  autoRun n [,m]        - Enable/disable WIPE program autorun. Enable if n=1.
                          Optional run delay is 0 < m < 255 seconds;
                          10 is default.
  program               - Start the WASP Interactive Programming Environment
  quiet                 - Don't display WASP commands being executed
  reset                 - Software reset
  save                  - Save parameters to EEPROM
  verbose               - Display WASP commands when running
  h                     - Print this help text
-----------------------------------
WASP slave network commands: (nodeid 255 = all nodes)
  netLed n, l, f, w     - For WASP slave n, configure its LED pixel strip
                          parameters:
                            l = # tricolor LED pixels (l > 0)
                            f = 8, for 800kHz, WS2812 LED drivers
                                4, for 400kHz, WS2811 drivers
                            c = 0, for RGB color wiring order (default)
                                1, for RBG
                                2, for GRB
                                3, for GBR
                                4, for BRG
                                5, for BGR
  netLedCtrl n, m       - Change the control pin for the LED pixel
                          strip of slave n to digital output pin #m
  netNodeId n, m        - Change the nodeid for slave n to m
  netPing n             - Query a WASP slave, group, or all
  netReset n            - Reset WASP slave, group, or all.
  netSave n             - Request a WASP slave, group, or all node to
                          save their configuration settings to EEPROM
Free RAM: 9197
Autorun [off]
>
```

Figure 10: WASP Controller - Command Console Help Screen

Note that, in the commands shown, the commas can be replaced with spaces.

autoRun Configure the autorun settings.
(Refer to section *4.1 Automatically Running a Lighting Effect*.)

program Enter WIPE mode for programming lighting effects. (Once in WIPE mode, the WIPE **exit** command returns back to command console mode.

quiet *This command is intended for expert users who modify WASP firmware code*. It undoes the effect of the **verbose** command.

reset Reset the WASP Lighting Controller.
If there is an unknown problem on the unit, the command may not work.
If the unit doesn't response to the reset command, power cycle the unit.

save Save the autorun settings to EEPROM memory to make them permanent.
(Refer to the **autoRun** command.)

verbose *This command is intended for expert users who modify WASP firmware code*. The command causes WASP network protocol command codes to be displayed on the console while in either the command console mode or WIPE mode.

h Display the summary of commands available in command console mode.

netLed Modify the pixel string configuration parameters on a specific WASP Lighting Slave. (Refer to section *3.1.2 Configuring the WASP Slave Units*.) Use the **netSave** command to save the change permanently. You should also reset the slave (**netReset**) and query it (**netPing**) to confirm permanence of the change.

netLedCtrl Modify the digital output pin,on the embedded microcontroller of a specific WASP Lighting Slave. (Refer to section *3.1.2 Configuring the WASP Slave Units*.) Use the **netSave** command to save the change permanently. You should also reset the slave (**netReset**) and query it (**netPing**) to confirm permanence of the change.

netNodeId Modify the WASP node ID associated with a specific WASP Lighting Slave. (Refer to section *3.1.2 Configuring the WASP Slave Units*.) Power cycle the slave after running this command, and query it using the **netPing** command to confirm the change.

netPing Query for the presence of a WASP Lighting Slave (or all slaves) on the WASP network[4]. The slave(s) respond with their WASP node ID, their configuration data, and version information.The information returned by the slaves is shown in *Figure 11*. The argument to this command specifies the slave's node ID. A value of 255 requests responses from all slaves.

```
> netPing 255
[netPing 255]
Node #2==>  F/W: 3  S/W: 5.00  Pin: 16  (50 pixels @ 400kHz, LED order #4)
Node #3==>  F/W: 3  S/W: 5.00  Pin: 16  (50 pixels @ 400kHz, LED order #4)
Node #4==>  F/W: 3  S/W: 5.00  Pin: 16  (50 pixels @ 400kHz, LED order #4)
=> PING done
>
```

Figure 11: WASP Controller - Sample of netPing Results

4 If a slave group has been defined (refer to section 4.3.2.3 The WIPE Functions), this command can be applied to that subset of slaves by specifying the group ID as the command's argument.

netReset Request a specific WASP Lighting Slave (or all slaves) to reset itself (themselves)[4]. The argument to this command specifies the slave's node ID. A value of 255 requests all slaves to reset themselves.

If there is an unknown problem on a slave, this command may not effect a reset. If a slave doesn't respond to this command, power cycle it.

netSave Request a specific WASP Lighting Slave (or all slaves) on the WASP network to save its (their) configuration data to EEPROM memory to make changes permanent[4].

4.3 Programming Lighting Effects on the Controller

This section describes how to program lighting effects using the WASP Interactive Programming Environment (WIPE). The controller can be configured to automatically run a specific program stored in its EEPROM memory.

4.3.1 First Program: WASP Hello World

This section provides the steps for creating a simple WIPE program and executing it on the WASP network. The progam can be used for basic testing of WASP Lighting Slave units.

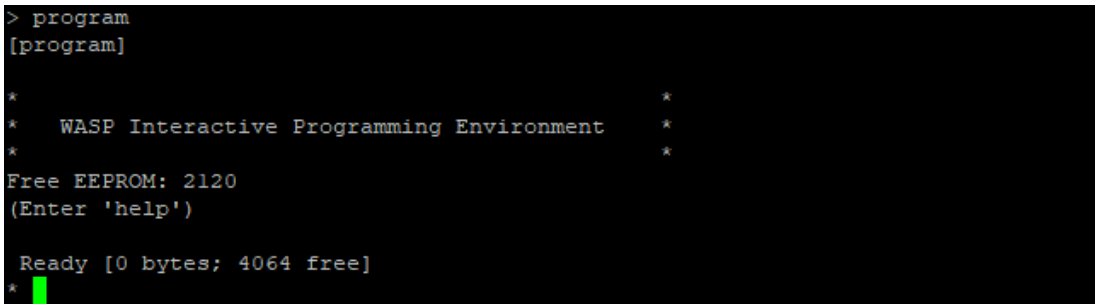
(1) Refer to section *4.1 Automatically Running a Lighting Effect* to power up the WASP Lighting Controller and disable the autoRun feature.

(2) Power cycle the controller.

(3) In the command console, enter:

```
program
```

The following response is displayed:



While in WIPE mode, and as long as a program isn't running, you can enter the following command to get a summary of WIPE commands, WIPE programming language statements, and WIPE functions:

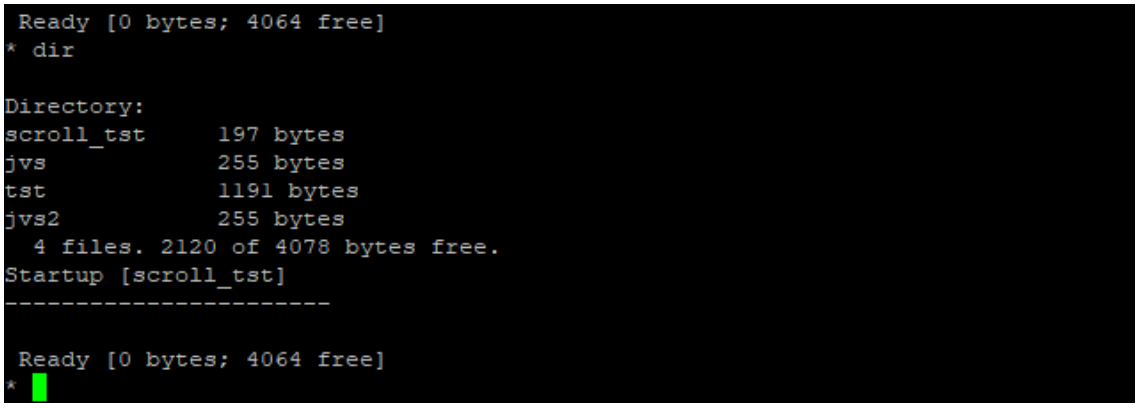
```
help
```

The help command is covered more fully in section 4.3.2.1 WIPE Commands.

(4) In the WIPE console, enter the following WIPE command:

```
dir
```

If there are programs already stored in memory, the response may look something like the following:



The line "`4 files. 2120 of 4078 bytes free.`" indicates that there are 2120 bytes of memory still available in EEPROM where WIPE programs are stored.

(5) If there are programs already in EEPROM and you wish to erase them, or if you find that there isn't enough room for this program, enter the following WIPE commands:

`dir`	(to display the list of programs)
`erase` ***filename***	(where *filename* is the name of a program, e.g. `scroll_tst`)

The commands can be repeated for additional program erasures.

(6) Now enter the following text, verbatim, in the console:

```
10 Reset(255)
20 Pause(0,1,0)
30 Bkgrd(255,255,0,0)
40 Pause(0,10,0)
50 Bkgrd(255,0,255,0)
60 Pause(0,10,0)
70 Bkgrd(255,0,0,255)
80 Pause(0,10,0)
90 Reset(255)
```

This WIPE program does the following (program line numbers are shown in the left margin in parentheses):

(10) Requests all WASP Lighting Slaves to reset themselves and
(20) Wait for 1 second before continuing.
(30) Set all pixels (the background colour) on all slaves to full brightness red.
(40) Hold the red colour for 10 seconds.
(50) Set the background colour on all slaves to full brightness green.
(60) Hold the green colour for 10 seconds.

(70) Set the background colour on all slaves to full brightness blue.
(80) Hold the blue colour for 10 seconds.
(90) Request all slaves to reset themselves; all pixels will turn off.

(7) If you wish, you may list the program and then test run it by entering the following sequence of WIPE commands:

```
list
run
```

(8) Save the program to EEPROM by entering the following WIPE command:

```
save tst
```

(9) Enter the following WIPE command to confirm that the program was saved:

```
dir
```

Program "tst" should now be listed in the directory.

(10) Set program "tst" as the program that will be autorun when the feature is subsequently enabled by entering the following WIPE command:

```
start tst
```

If you enter the **dir** command again, you will see the following line near the bottom of the output which confirms that "tst" is the program to be autorum:

```
Startup [tst]
```

(11) Enter the following WIPE command to leave WIPE mode and return to command console mode:

```
exit
```

(12) Enter the following sequence of commands in the command console to enable autorun with a delay of 15 seconds following power up or reset before program "tst" starts running:

```
autoRun 1, 15        (Enable autorun with 15 second delay)
save                 (Save the autorun settings to make them permanent)
```

(13) Now you can reset or power cycle the WASP Lighting Controller to run the test. If the WASP Lighting Slaves are correctly connected and configured, after 15 seconds the colour sequence red, green, blue is displayed on all pixel strings, with each colour being held for 10 seconds. Ten seconds after blue is displayed, all pixel strings go black.

4.3.2 The WASP Interactive Programming Environment (WIPE)

This section describes the WASP Interactive Programming Environment (WIPE), the commands available while the WASP Lighting Controller is in WIPE mode, the WIPE programming language, and the WIPE functions (most of which control the WASP Lighting Slaves). Lighting effects can be created by writing WIPE programs which can be stored permanently on the controller.

While the controller is in command console mode, entering the following command takes you to WIPE mode:

```
program
```

The command prompt * indicates that the controller is in WIPE mode. (The command prompt > indicates command console mode; refer to section *4.2 Command Console Commands*.)

When WIPE mode is entered, the controller's indicator LED turns solid green and output similar to *Figure 12* is seen on the console. The line "`Free EEPROM: 2120`" indicates how many bytes of program storage is still available.

The line "`Ready [0 bytes; 4064]`" indicates that the current program occupies 0 bytes of RAM memory (since a program has been entered or loaded yet) and that 4064 bytes of RAM are available for entering a program. This upper bound represents the program storage capacity of the EEPROM memory. No single program larger than this can be stored permanently in EEPROM. With a program of this size, if there are any programs already stored in EEPROM, they would have to be erased before this program could be saved.

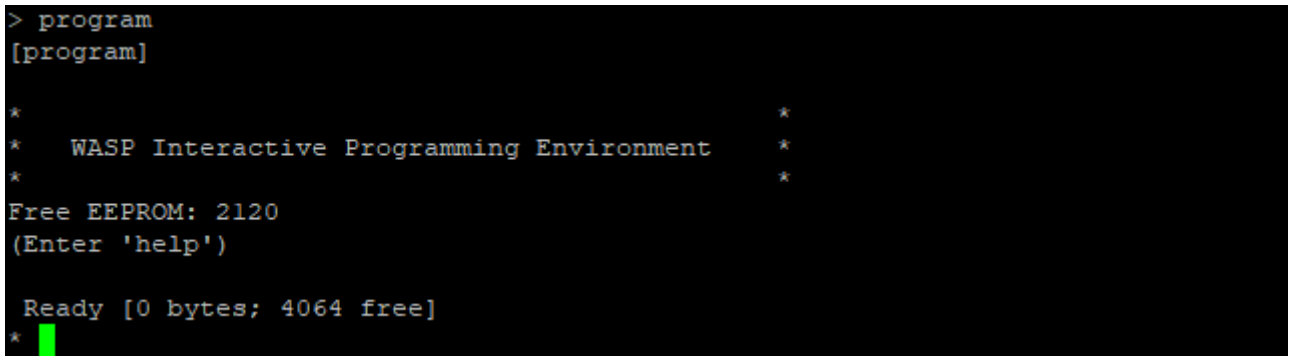


Figure 12: WIPE Startup Screen

A quick reference of WIPE commands, WIPE programming language statements, and WIPE programming language functions can be displayed by entering the following WIPE command:

```
help
```

The command's output is shown in *Figure 13* and *Figure 14*.

The next section describes the commands available in WIPE mode.

```
 Ready [0 bytes; 4064 free]
* help


** WIPE Help **

WIPE is BASIC-like and case sensitive
- To edit a line, use same line #

COMMANDS
del <m> [- n]    - Delete line m thru n
dir              - List saved programs
erase <name>     - Delete program
exit             - Exit WIPE
help             - Show help
load <name>      - Load program
list m [- n]     - List lines m thru n
prof             - Show all symbols
renum            - Renumber lines evenly from 5 - 250
run              - Run program; 'x' to quit.
save <name>      - Save program
start <name>     - Set startup program

STATEMENTS
if <expr>               Skip next line if <expr> false
label <id>
let <id> = <expr>
goto <label-id>
print <expr>
var <id> : <type>       in {int, float, bool}
<func>(<params>)  (see FUNCTIONS)

  where: <expr> is with no precedence

... <Enter>
```

Figure 13: WIPE Help Screen (part 1)

```
... <Enter>


FUNCTIONS
GENERAL
        Pause(min,sec,100ths)
WASP GENERIC
        Bkgrd(dst,r,g,b)
        Group(dst,grp,l,r)
        Line(dst,r,g,b,start,len)
        Reset(dst)
        Shift(dst,shift)
        State(dst,opts),        opts is OR'd collection of
                                {SAVE(1), RESTORE(2), RESUME(4), SUSPEND(8)}
        Swap(dst,r,g,b,r',g',b')

SPECIAL FX
        Rain(dst,start)
        Cycle(dst)
        Speed(dst, dly) FX speed
        Twinkle(dst,minDly,maxDly,burst,hold)

 Ready [0 bytes; 4064 free]
*
```

Figure 14: WIPE Help Screen (part 2)

4.3.2.1 ***WIPE Commands***

This section describes the WIPE environment commands as follows:

exit Exit WIPE and return to command console mode.

help Display a quick reference summary of WIPE commands, and WIPE programming language statements and functions. The output was shown in *Figure 13* and *Figure 14*.

File Oriented Commands:

save Save the program that is currently in RAM memory to persistent EEPROM memory using the given name.

e.g. `* save program1`

dir List the programs that are stored in EEPROM.

e.g. `* dir`

```
Directory:
program1       197 bytes
program2       255 bytes
program3       1191 bytes
jvs2           255 bytes
  4 files. 2387 of 4078 bytes free.
Startup [program2]
----------------------

Ready [0 bytes; 4064 free]
```

load Load the named program (stored in EEPROM) into RAM memory so that it can be edited and run.

e.g. `* load program2`

erase Delete the named program from EEPROM memory.

e.g. `* erase program3`

start Configure the named program (stored in EEPROM) for autorun.

e.g. `* start program2`

Program names must be no greater than 12 characters in length.

Program Oriented Commands:

run Run the program that is currently in RAM memory. (You can either enter a program (see section *4.3.2.2 The WIPE Programming Language*) or load a previously entered program that is stored in EEPROM memory (refer to the **load** command). While a program is running, execution can be terminated by entering **x** in the console.

list List the lines of the program that is currently in RAM memory.

e.g. List all program lines:

```
* list
```

List lines numbered 15 thru 100, inclusive:
```
* list 15 - 100
```

List lines numbered up to 150:
```
* list - 150
```

List lines numbered from 5 onward:
```
* list 5 -
```

renum Automatically renumber the program lines. (This is useful when you need to create a gap between two consecutively numbered lines.)

del Delete program lines.
e.g. Delete all program lines:
```
* del 1 - 255
```

Delete program lines numbered 15 thru 100, inclusive:
```
* del 15 - 100
```

Delete lines numbered up to 150:
```
* del - 150
```

Delete lines numbered from 5 onward:
```
* del 5 -
```

prof Show (profile) all symbols (variables and labels) currently defined along with their values. (Label values are internally generated when a program runs and are of no user significance.)

4.3.2.2 The WIPE Programming Language

WIPE implements a very simple interpreted programming language (simply referred to as WIPE). Statements and functions can be entered directly at the command prompt for immediate execution. This is especially useful for getting immediate feedback on functions, particularly those for which you wish to tune pixel colours. While the slaves are powered up and the network correctly configured (refer to section *3 WASP Network Setup, Configuration, and Administration*), they will respond to the commands transmitted by the controller in response to WIPE functions being executed.

By prefixing a WIPE statement with a line number, the statement is entered in the program stored in RAM memory. Use WIPE command, **list**, to list the program lines. To edit an existing line, enter the line number followed by the new statement.

Given the limited amount of EEPROM memory available on the controller (4 kilobytes), there are a number of constraints placed on programs to conserve memory as much as possible:

- Variable names and label names are limited to 4 characters. They must start with a letter.
- Program line numbers are limited to the range of 1 thru 255, inclusive. (The WIPE **renum** command can be used create a gap for new line insertions between consecutively numbered lines.)

Program statement keywords, function names, variable names and labels are all case-sensitive.

Data types supported by WIPE are:

- **int** 32-bit integers (value range of -2,147,483,648 to 2,147,483,647).
- **float** 32-bit floating point numbers (value range of $-3.4028235 \times 10^{38}$ to 3.4028235×10^{38}). They have a precision of 7 decimal digits.
- **bool** Boolean values 0 or 1 (equivalent to false and true, respectively).

A maximum number of 200 variables and labels can be defined.

The following are the WIPE programming language statements:

`var` <id> **`:`** <type>	Declare a variable named by <id> of type <type>. (The spaces around the colon are required.)
`label` <id>	Declare a label (named by <id>) that can be used as a target address for a **goto** statement.
`let` <id> **`=`** <expr>	Define the value of the variable named by <id>. <expr> is an expression and is defined below. (The spaces around the equal sign are required.)
`goto` <id>	Jump to the program line where the label, named by <id>, is declared. (Refer to the **label** statement.)
`if` <expr>	Conditional execution. <expr> is a boolean expression (defined below) which must evaluate to either 1 (true) or 0 (false). If the expression evaluates to true, execute the next statement; otherwise, skip the next statement and resume execution at the statement that follows. Typically, the next statement would be a **goto**.
`print` <expr>	Output information to the controller's console. This is helpful for debugging programs. The expression, <expr>, is either a double quoted character string or a variable name. Subsequent print statements normally continue on the same screen line. A quoted character string can include the special character sequence, **`\n`**, to force a new line to start. If <expr> is a variable name, the current value of the variable is output.
<func> **`(`**<params>**`)`**	Call a WIPE function named by <func> with specified parameters. <param> is a comma separated list of expressions that are defined below. The WIPE functions are described in section *4.3.2.3 The WIPE Functions*.

WIPE Expressions

Expressions in WIPE are arithmetic expressions that take the following forms:
(Note that operators all have equal precedence. Operands are evaluated left-to-right. There are no parentheses to change evaluation order.)

<expr> := <id>	An expression can be a variable name.
<expr> := <const>	An expression can be a literal constant value such as 0, 1, 115, and 2.23. The value must be consistent with the expression's type.
Numeric expressions:	
<expr> := <expr> **+** <expr>	An addition is an expression. The + operator takes two expression operands.
<expr> := <expr> **−** <expr>	A subtraction is an expression. The − operator takes two expression operands.
<expr> := **−**<expr>	Negation is an expression. The − operator can also take just one argument to negate the operand's value.
<expr> := <expr> ***** <expr>	A multiplication is an expression. The * operator takes two expression operands.
<expr> := <expr> **/** <expr>	A division is an expression. The / operator takes two expression operands. For an integer type expression, the result of `a / b` is the integer portion of `a` divided by `b`; the remainder is dropeed.
<expr> := <expr> **%** <expr>	A modulo is an expression. The `%` operator takes two expression operands. The operator is valid only for integer type expressions. The result of `a % b` is the integer remainder of `a` divided by `b`.
Boolean expressions:	
<expr> := **!**<expr>	Boolean complement is an expression. The `!` operator takes a single boolean operand. The operator results in value 1 (true) if the operand's value is 0 (false); otherwise, it results in value 0 (false).
<expr> := <expr> **=** <expr>	Equality is an expression. The = operator takes two expression operands. The operator returns a boolean value of 1 (true) if both operands evaluate to the same value; 0 (false), otherwise.
<expr> := <expr> **!=** <expr>	Inequality is an expression. The != operator takes two expression operands. The operator returns a boolean value of 0 (false) if both operands evaluate to the same value; 1 (true), otherwise.
<expr> := <expr> **<=** <expr> **<** **>=** **>**	Less-than-or-equal-to is an expression. The <= operator takes two expression operands. The operator returns a boolean value of 1 (true) if the left operand's value is no greater than than of the right operand's value; 0 (false), otherwise. The less-than (<), greater-than-or-equal (>=), and greater-than (>) operators work

similarly.

<expr> := <expr> **&&** <expr>	A logical AND is an expression.The `&&` operator takes two boolean expression operands and returns a boolean value of 1 (true) if both operands evaluate to 1 (true); 0 (false), otherwise.
<expr> := <expr> **||** <expr>	A logical OR is an expression.The `||` operator takes two boolean operands arguments and returns a boolean value of 1 (true) if either operands evaluates to 1 (true); 0 (false), otherwise.

4.3.2.3 The WIPE Functions

The following functions are defined in the WIPE programming language:
(All parameters are expressions with a value range of either 0 to 255 or -128 to 127.)

General Functions:

`Reset(`dst**`)`**
Reset the given destination, which can be a specific WASP slave node ID (dst in {2, 3, ..., 6}), all slaves (dst = 255), or a slave group (dst in {128, 129, ..., 132}). (To define a slave group, refer to the **Group** function.)

`Pause(`min`, `** sec`, `** hundredths**`)`**
Pause program execution for specified number of minutes, seconds, and hundredths of a second.

WASP Generic Graphics Functions:

`Bkgrd(`dst`, `** red`, `** green`, `** blue**`)`**
Set all pixels to the specified colour (parameters red, green, and blue) for the given destination (parameter dst). The destination can be a specific WASP slave node Id (dst set to a value of 2 thru 6), all slaves (dst set to 255), or a defined group of slaves (dst set to a value of 128 thru 132). To define a slave group, refer to the **Group** function.

`Group(`dst`, `** group`, `** left`, `** right**`)`**
Define a specific slave ID as a member (parameter dst) of a specified group (parameter group) of WASP slaves. A group can have from 1 to 5 members and must be defined as a ring. That is, each member has one unique slave as a neighbor to its left and one unique neighbor to its right. For example:

- For a single-slave group #128 and having slave #4 as its sole member, you would call `Group(4, 128, 4, 4)` since the slave completes its own ring.
- For a two-slave group #132, having slaves #2 and #3 as its members, you would call `Group(2, 132, 3, 3)` and `Group(3, 132, 2, 2)`.

Grouping slaves together is useful when using the **Shift** function.

`Line(`dst`, `** red`, `** green`, `** blue`, `** start`, `** length**`)`**
Draw a line at a given destination (parameter dst) by setting a number of pixels (parameter length) to the specified colour (parameters red, green, and blue) starting at a specified pixel number (parameter start). The destination can be a specific WASP slave

node Id (dst in {2, 3, ..., 6}, all slaves (dst = 255), or a slave group (dst in {128, 129, ..., 132}). To define a slave group, refer to the **Group** function. Pixels are numbered from 0 onward; with a pixel string length configured to *N*, the highest numbered pixel is (*N* – 1). (Refer to section *3.1.2 Configuring the WASP Slave Units* for information on configuring a pixel strings.) The value, start + length, should not exceed the highest pixel number as configured for the string.

`Shift(`dst**`,`** shift**`)`**

Shift the pixel colours for the given destination slave group (parameter dst in {128, 129, ..., 132}) by the specified number of pixels (parameter shift). (To define a slave group, refer to the **Group** function.) Parameter shift can range from -20 to 20. A negative value causes each member to shift its pixels to its left neighbor, while a positive value causes each member to shift its pixels to its right neighbor.

`State(`dst**`,`** options**`)`**

Affect the pixel-drawing state (parameter options) of the specified destination (parameter dst). The destination can be a specific WASP slave node Id (dst in {2, 3, ..., 6}, all slaves (dst = 255), or a slave group (dst in {128, 129, ..., 132}). (To define a slave group, refer to the **Group** function.) The options parameter can take on the following values:

1	SAVE the current pixel colours in memory.
2	RESTORE the saved pixel colours that were last saved.
4	RESUME pixel updates. (This option undoes the SUSPEND option.)
8	SUSPEND the refreshing of pixels. The destination slaves delay their visible response to graphic and specical effects functions until the RESUME option is subsequently specified. By suspending pixel updates, you can freeze their current colours, call a sequence of graphics/effects functions, then use RESUME to get the slave(s) to produce the final result.

Also the following combinations:

5	= 1 + 4	Both SAVE and RESUME.
9	= 1 + 8	Both SAVE and SUSPEND.
6	= 2 + 4	RESTORE first, then RESUME.
10	= 2 + 8	RESTORE first, then SUSPEND.

`Swap(`dst**`,`** old_red**`,`** old_green**`,`** old_blue**`,`** new_red**`,`** new_green**`,`** new_blue**`)`**

Swap one colour of pixel (parameters old_red, etc) for a new colour (parameters new_red, etc.) at a given destination. The destination can be a specific WASP slave node Id (dst in {2, 3, ..., 6}, all slaves (dst = 255), or a slave group (dst in {128, 129, ..., 132}). (To define a slave group, refer to the **Group** function.) All pixels of the same old colour are replaced by the new colour.

WASP Animated Special Effects Functions:
(The animation speed of the effects is controlled by the **Speed** function. In each function, the destination can be a specific WASP slave node Id (dst in {2, 3, ..., 6}, all slaves (dst = 255), or a slave group (dst in {128, 129, ..., 132}). To define a slave group, refer to the **Group** function.)

`Rain(`dst**`,`** start**`)`**
Run a rainbow effect on the given destination with the start of the rainbow beginning at the specified pixel number (parameter start). Pixels are numbered from 0 onward; with a pixel string length configured to *N*, the highest numbered pixel is (*N* – 1). (Refer to section *3.1.2 Configuring the WASP Slave Units* for information on configuring a pixel strings.)

`Cycle(`dst**`)`**
Run a cycling rainbow effect on the given destination.

`Twinkle(`dst**`,`** min_delay**`,`** max_delay**`,`** burst_size**`,`** hold_time**`)`**
Run a twinkling effect on the given destination. Before calling this function define the background pattern using a combination of the generic graphic functions. Twinkles are rendered in full brightness white colour. The effect is more pronounced when the background pattern is less bright. Twinkles occur on a random subset of pixels, the subset size being limited by the maximum burst size parameter (burst_size). The twinkles occur with a random delay between bursts, the relative delay range being constrained by parameters min_delay and max_delay. Twinkle bursts have a relative hold time defined by parameter hold_time. The relative timings are affected by the effect speed set by function **Speed**.

`Speed(`dst**`,`** delay**`)`**
Adjust the animation speed of the special effects on a given destination. Generally, the larger the value, the slower the effect. Parameter, delay, defines the relative speed with the following special values:

0	Pause the effect indefinitely.
1	Advance the effect by a single step. The effect pauses indefinitely between each subsequent call of `Speed(`*x*`, 1)`.

5 Firmware Installation

This section describes how to update the firmware in WASP Lighting Controller and WASP Lighting Slave units. The information is intended for development and support purposes. It is not intended for those who operate the units on a WASP network.

5.1 Installing Firmware on a WASP Slave Unit

5.1.1 First-Time Firmware Installation Using an FTDI Adapter

To install firmware in a WASP Lighting Slave unit for the first time, the following is an overview of actions that are needed and which are expanded upon below:

- Obtain the WASP Lighting Slave source code—an Arduino sketch named **WASP_Slave_v*aa.bb*,** where *aa.bb* identifies the slave firmware version.
- Obtain an FTDI USB-to-Serial bridge adapter board.
- Install the Arduino Integrated Development Environment (IDE) software.
- Install the Moteino board support package and libraries into the Arduino IDE.
- Connect the FTDI adapter to the computer and Moteino board.
- Compile and upload to the Moteino board the WASP Lighting Slave source code.

The WASP Lighting Slave source code is located at **<??TBD>**. The source code folder will need to be copied into the folder where Arduino sketches are stored after having installed the IDE and Moteino board support package and libraries.

Next, it is necessary to download and install the Arduino IDE software (downloadable from https://www.arduino.cc). You will then need to install the Moteino package in your Arduino IDE—this includes definitions for all Moteino boards. First add the Moteino core json definition URL to your Board Manager. In the Arduino IDE, select **Files > Preferences** (that is, select "Preferences" under the "File" menu), and in the "Additional Boards Manager URLs" add "https://lowpowerlab.github.io/MoteinoCore/package_LowPowerLab_index.json" as shown in *Figure 15*.

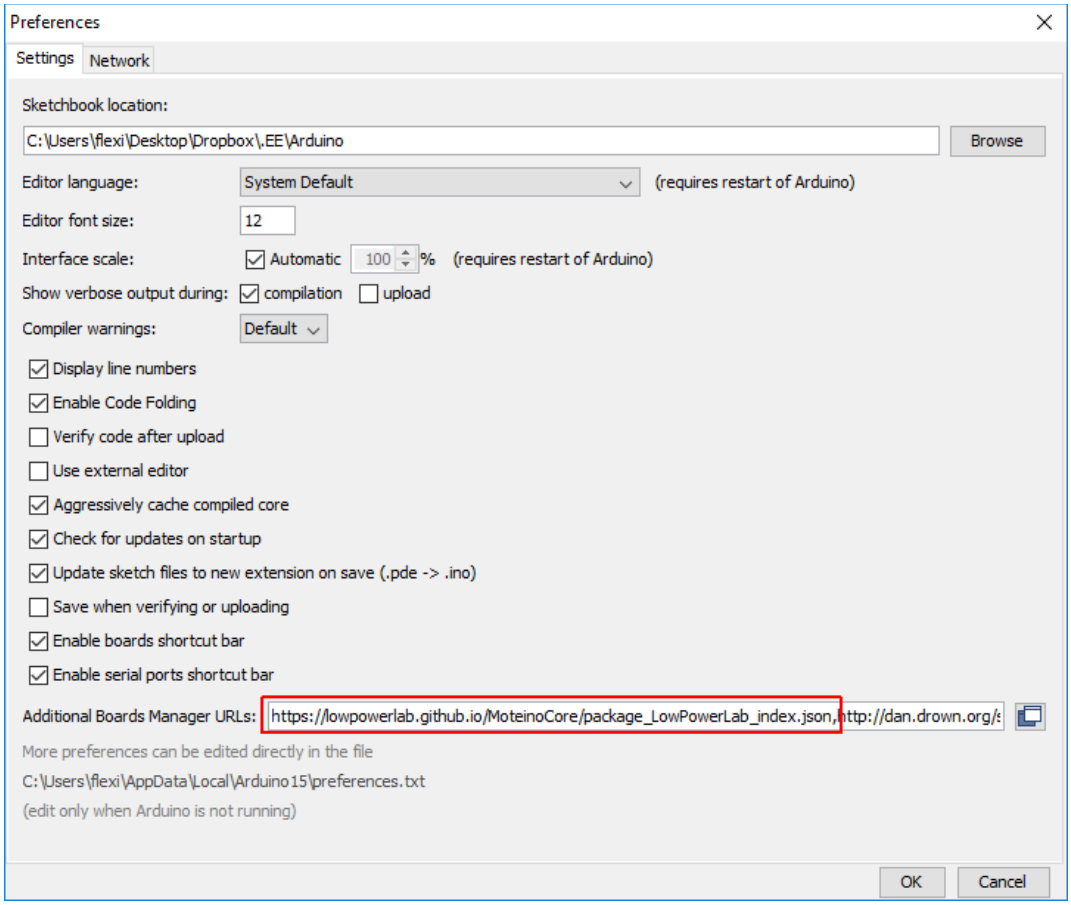


Figure 15: Arduino Preferences

Now select **Tools > Board:... > Boards Manager...** to open up the Boards Manger window. For the AVR based Moteino and MoteinoMEGA boards you need to install the **Moteino AVR Boards by LowPowerLab** package as shown in *Figure 16*.

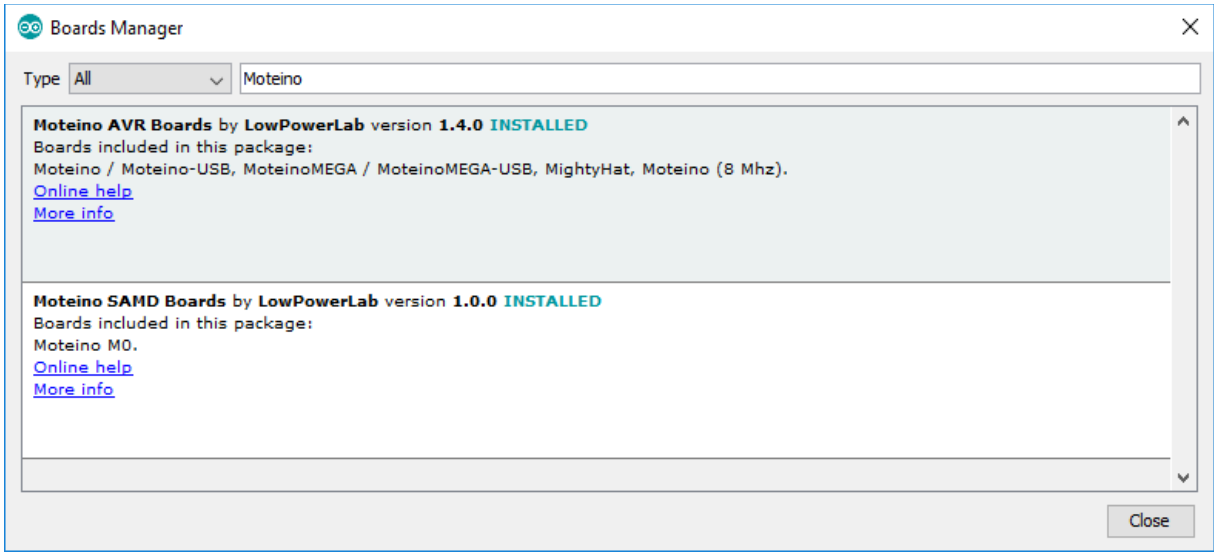


Figure 16: Arduino Board Package Updater

Once installed the new boards will show up in your **Tools > Boards** menu under section **Moteino AVR Boards** as shown in figure *Figure 17*.

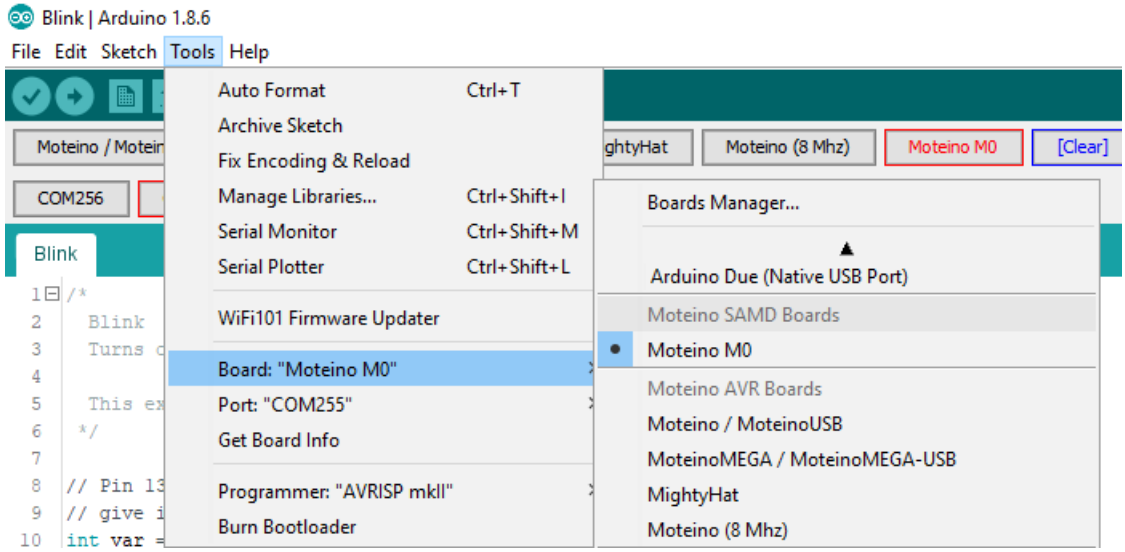


Figure 17: Moteino AVR Boards in Arduino's Board Menu

You will next need to install the Moteino libraries from the LowPowerLab GitHub repository (https://github.com/lowpowerlab). One thing that is important is to download the repositories as ZIP files, then rename their contents (remove the "-master" in the folder name), and then put them in the **libraries** subfolder where your Arduino sketches are stored (specified in your Arduino preferences). You will need the RFM69 and SPIFlash libraries. The Arduino Web site has a library installation tutorial (https://www.arduino.cc/en/Guide/Libraries).

Before proceeding further be sure to have the WASP Lighting Slave source code stored in the Arduino **sketches** folder. You should quit and restart the Arduino IDE so that all IDE related changes take effect.

Obtain an FTDI USB-to-Serial bridge adapter such as one pictured in *Figure 18*. (Note the pin assignments on the black connector, with DTR on the left and ground on the right.) Connect the adapter to the slave's embedded Moteino board, as shown in *Figure 19*, being sure to align the DTR pins and ground pins together.

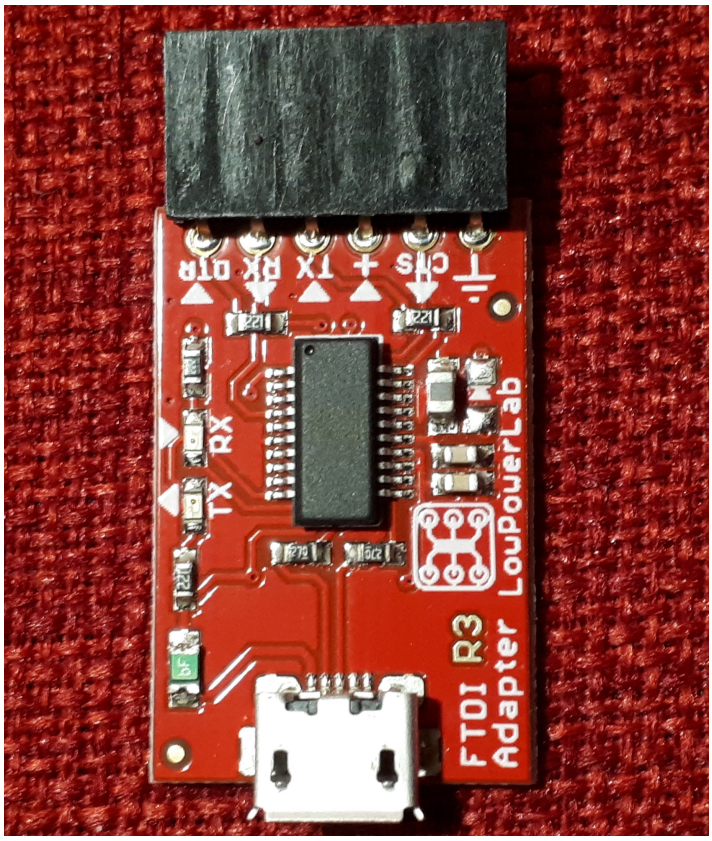


Figure 18: FTDI USB-to-Serial Bridge Adapter

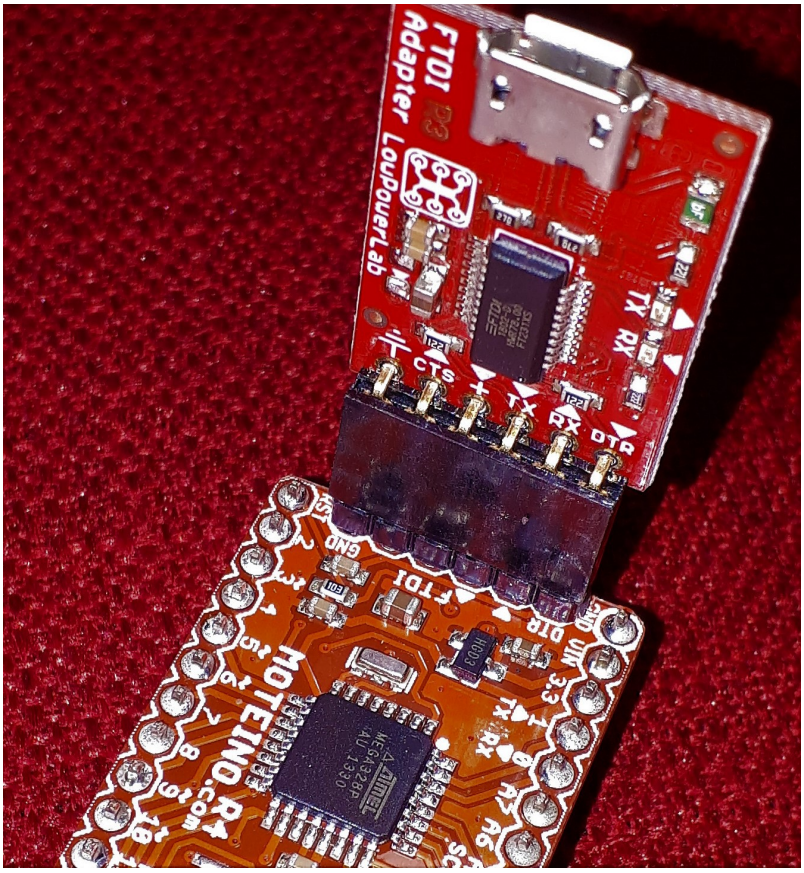


Figure 19: FTDI Adapter Connected to Moteino

You can now compile and upload the slave software:

1. In the Arduino IDE, select **File > sketchbook >** <*source code folder*> **>** ... **>** **WASP_Slave_v*aa.bb*** to load the software into the IDE's editor.
2. Select **Tools > Board:** and, from the menu that opens, select the appropriate Moteino board type.
3. Select **Tools > Port** and, from the menu that opens, select the Serial (COM) port to which the FTDI adapter is connected.
4. Select **Sketch > Verify/Compile** to compile the software.
5. Finally, select **Sketch > Upload** to upload the compiled software to the Moteino board.

You can export the compiled binary file for subsequent wireless updates to the WASP Lighting Slave unit (refer to section *5.1.2 Wireless Updates with a Moteino Wireless Programmer*). To do so, in the Arduino IDE, select **Sketch > Export compiled Binary**. The resulting Update file is stored in the same folder as the WASP source code folder and is named either **WASP_Slave_v*aa.bb*.ino.Moteino.hex** (for a Moteino board) or **WASP_Slave_v*aa.bb*.ino.MoteinoMEGA.hex** (for a MoteinoMEGA board), where *aa.bb* is the firmware version identifier.

5.1.2 **Wireless Updates with a Moteino Wireless Programmer**

Normally, updates to a WASP Lighting Slave unit can be accomplished wirelessly using a Moteino Wireless Programmer unit as pictured in Figure 20. The following steps assume that an Update file is available. (To produce an update file, follow the steps in section *5.1.1 First-Time Firmware Installation Using an FTDI Adapter*, including the last step for exporting the compiled binary file.)

Figure 20: Moteino Wireless Programmer Unit

The following are the steps for wireless update:

1. Obtain an appropriate Update file—**WASP_Slave_v*aa.bb*.ino.Moteino.hex** (for a Moteino board) or **WASP_Slave_v*aa.bb*.ino.MoteinoMEGA.hex** (for a MoteinoMEGA board), where *aa.bb* is the firmware version identifier. Note the folder where the file is stored.
2. Ensure that the Moteino Wireless Programmer unit is frequency-compatible with the WASP Lighting Slave unit (433 MHz).

   *The Moteino Wireless Programmer unit is basically a Moteino-USB board that runs a special Arduino sketch. (The sketch is located in the same Arduino folder as the sketches for the WASP controllers and slaves. The sketch is named **WASP_Node_Programmer_vaa.bb**, where aa.bb is the version identifier.) Note that the unit operates at a baud rate of 115200 when connected to a computer's USB port. (So, the serial port settings are 115200 baud, 8 data bits, no stop bit, 1 parity bit.)*
3. Obtain the Wireless Programming application for Windows (available from LowPowerLab.com). The program does not require installation and can be run from a flash drive if desired.

The program can also be found under .../Arduino sketchbook/Xmas_Pixel_lighting/Wireless_Programming_Support_Files/

4. Ensure that there is no WASP Lighting Controller unit powered up since it could interfere with the wireless update process.
5. Power up the WASP Lighting Slave unit. Note its slave ID number.
6. Connect the Moteino Wireless Programmer unit to the Windows PC's USB port. Note the Serial (COM) port to which it is connected.
7. Start up the Wireless Programming application (**WirelessProgramming.exe**).
8. In the application:
   - Click the "Browse Hex file.." button and browse to, and select, the Update file.
   - In the “Serial port of programmer Moteino (baud 115200):” field, select the COM port to which the Moteino Wireless Programmer unit is connected.
   - In the “Target node ID:” field, select the WASP node # for the WASP Lighting Slave unit to be reprogrammed.
   - Leave the “HEX lines per RF packet:” field at the default of “3 (faster)” unless the programmer is having communication issues with the WASP slave.
   - Leave the “Run OTA.py:” field at the default value (unchecked).
9. In the WirelessProgramming application, click the "Start!" button. The "Log:" window should display the following:

```
Opening COM<n> @ 115200baud...          (where <n> is the COM port number)
SET TARGET: TO:<m>                      (where <m> is the slave node #)
Moteino: [TO:<m>:OK]                    (where <m> is the slave node #)
TARGET SET OK
FLX?
FLX?
Moteino: [FLX?OK]
HANDSHAKE OK!
TX > FLX:<#>:<...> RX > FLX:<#>:OK      (where <#> is sequence # starting at 0 and <...> is
                                         a hex code sequence)
  ...
TX > FLX:<#>:<...> RX > FLX:<#>:OK
Moteino: [FLX?OK]
EOF
SUCCESS! (time elapsed: <N>s)           (where <N> is an elapsed time)
```

The programming process takes about 30 seconds to complete.

5.2 Installing Firmware on a WASP Controller Unit

To install firmware in a WASP Lighting Controller unit, the following actions are needed:

1. Obtain the WASP Lighting Controller source code—an Arduino sketch named **WASP_Controller_v*aa.bb*,** where *aa.bb* identifies the controller firmware version.
2. Unlike for a WASP slave, an FTDI USB-to-Serial bridge adapter board is not required for a WASP controller since the controller's embedded Moteino board already has an onboard USB-to-Serial bridge.
3. Follow section *5.1.1 First-Time Firmware Installation Using an FTDI Adapter* to:
    - Install the Arduino IDE software.
    - Install the Moteino board support package and libraries into the Arduino IDE.
4. Connect the WASP Lighting Controller unit to the computer's USB port.
5. Follow section *5.1.1 First-Time Firmware Installation Using an FTDI Adapter* to compile and upload to the WASP controller the WASP Lighting Controller source code.