Duck Descriptions
- DuckLinks: 2
- MamaDucks: 3
- PapaDuck: 3

Raw Materials
- Software 4
- Hardware 4

Arduino Software Preparation
- Install Clusterduck Protocol 5
  - Using the Library Manager “COMING SOON!!!” 5
  - Importing as a .zip Library 5
  - Manual Install 6

Prepare the Firmware
- Install VCP Driver 6
- Install the necessary libraries that work with ClusterDuck Protocol: 6
  - Add ESP32 Board Library 6
  - Add Other Libraries 7
- Load the Heltec ESP32 Board to your Arduino IDE: 7

Install The firmware
- Connect ESP32 board to your computer via USB cable. 8
- In Arduino IDE, select Tools > “board” to Heltec_WiFi_LoRa_32 9
- In Arduino IDE, select Tools > “port” to <USB_PORT> 9
- Select Duck: 9
  - Choose an Example File 10
    - MamaDuckExample: Choose a DuckID and modify the code for your needs for api references and more explanation see attachment 1. 10
    - PaPaDuckExample: For the papa you need to setup a WiFi connection 10
- Upload the firmware 11

FAQs

API Reference And Quickstart ClusterDuck Protocol
Duck Descriptions

Ducks are IoT devices that connect together to form simple mesh networks. The Ducks utilize a combined network of LoRa (Long Range) technology, WiFi, Bluetooth, and sometimes other connectivities. When the Ducks need to communicate with each other they transmit over LoRa, a long-range and power efficient radio protocol. Often a user will need to communicate with the Ducks and may use WiFi. A networked cluster of Ducks - a ClusterDuck - is composed of several types of ducks: the DuckLink, MamaDuck, PapaDuck.

**DuckLinks:**

These are the basic nodes of the mesh network. The DuckLinks create a WiFi network where users can connect to it and submit emergencies. The DuckLink collects that data and transmits it to the MamaDuck using LoRa (915 MHz in the United States, 433 MHz in Europe and Asia). Anyone with a working WiFi device such as a smartphone or laptop can connect to a DuckLink.

**MamaDucks:**

MamaDucks act as central hubs to DuckLink groups. The MamaDuck is able to receive data over LoRa from the DuckLinks and transmit this data further into the network. This transmission can occur through other MamaDucks on the way towards the PapaDuck (once again using LoRa). The MamaDuck has most of the same properties as a Ducklink, though small changes in the device firmware help to optimize the architecture of the network.
PapaDuck:

PapaDuck is the final Duck in the ClusterDuck and transmits network data to the internet. When communicating with other Ducks the PapaDuck similarly uses LoRa. The data that the PapaDuck receives gets pushed to the OWL Data Management System (DMS), the cloud platform, through the Internet. It acts like a gateway that collects data from MamaDucks and then upload it to OWL.
Raw Materials

Software

You will need the Arduino IDE to install the firmware on your Arduino devices. The Ducklinks use our ClusterDuck Protocol. Project OWL’s open source Duck firmware. Depending on your computer setup, you may also need USB to UART drivers. If you have any problems, please reach out on the Project OWL Github or in the Project OWL slack.

Hardware

There are many combinations of electronics, batteries, and enclosures that can be used to build a Duck. The following are a recommended minimum set of materials commonly used to create a functional Duck:

LoRa Board: Heltec ESP32 WiFi + LoRa board, such as this: https://www.amazon.com/MakerFocus-Development-Bluetooth-0-96inch-Display/dp/B076MSLFC9/ref=sr_1_3

Battery: 3.7v 1.25 Micro JST battery, such as this: https://www.amazon.com/MakerFocus-Rechargable-Protection-Insulated-Development/dp/B07CXNQ3ZR/ref=sr_1_2

Data Transfer Cable: USB Micro connector to USB for your computer, such as this: https://www.amazon.com/AmazonBasics-Male-Micro-Cable-Black/dp/B072J1BSV6/ref=sr_1_3

Enclosure: Plastic Box (or 3D print, or rubber ducky, or anything you want really) if a case is necessary, this would work: https://www.amazon.com/gp/product/B07DVS1HC4/ref=ppx_yo_dt_b_asin_title_o06_s00?ie=UTF8&psc=1
Arduino Software Preparation

Install Clusterduck Protocol

Using the Library Manager “COMING SOON!!!”

Importing as a .zip Library

In the Arduino IDE, navigate to Sketch > Include Library > Add .ZIP Library. At the top of the drop down list, select the option to "Add .ZIP Library".

Navigate to the downloaded ClusterDuck Protocol Folder and select. Return to the Sketch > Include Library menu. You should now see the library at the bottom of the drop-down menu. It is ready to be used in your sketch. The zip file will have been expanded in the libraries folder in your Arduino sketches directory.

Manual Install

You can also copy the library manually into your Arduino libraries folder as follows.

1. Copy ClusterDuck folder (Download Zip or Clone to local machine)
2. Navigate to your Arduino folder. This can be found in your default Documents folder.
3. Navigate to the library folder
4. Paste into library folder
5. Restart Arduino
6. You should now be able to see examples by going to File -> Examples -> ClusterDuck

You should be able pull new commits directly to this folder in your Arduino library.

**Prepare the Firmware**

Open Arduino IDE. If you do not have this developer environment yet, download the Arduino IDE here: [https://www.arduino.cc/en/main/software](https://www.arduino.cc/en/main/software)


**Install VCP Driver**

- If MacOS:
  - Make sure you click on the folder that says Legacy MacVCP Driver and then click on ‘Silicon Labs VCP Driver.pkg’
  - Once it is finished installing, go to Mac System Preferences -> Security and Privacy -> General. Make sure Silicon Labs is allowed.

**Install the necessary libraries that work with ClusterDuck Protocol:**

**Add ESP32 Board Library**

1. Open Arduino
2. Go to Preferences, and in the “Additional Boards Manager URLs” please add the following string:

Add Other Libraries

Go to Sketch > Include Library > Manage Libraries:

Search for LoRa and install “LoRa by Sandeep Mistry”
Search for u8g2 and install u8g2

Include following libraries as well:
Download and install Zip

https://www.arduinolibraries.info/libraries/arduino-timer
https://github.com/espressif/arduino-esp32

Load the Heltec ESP32 Board to your Arduino IDE:

In Arduino IDE, select Tools > Board > Boards Manager, and in the pop up window type “esp32” into the search field. You should see the “esp32 by Espressif Systems” library. Install this library.

Once the board library is installed, you are ready to use the Arduino IDE.
Install The firmware

DuckLink Hardware and Firmware Assembly

1. Connect ESP32 board to your computer via USB cable.
2. In Arduino IDE, select Tools > “board” to Heltec_WiFi_LoRa_32

3. In Arduino IDE, select Tools > “port” to <USB_PORT>

4. Select Duck:

At this point you will need to choose what kind of Duck you want to make. There are Examples for different Ducks included in the ClusterDuck Protocol. Go to File -> Examples -> Clusterduck.
Choose an Example File

*Note: every Clusterduck network needs at least 1 Mama and 1 Papa*

**MamaDuckExample:** Choose a DuckID and modify the code for your needs for api references and more explanation see down below or go to [GitHub](https://github.com/)

**PaPaDuckExample:** For the papa you need to setup a WiFi connection

```
#define SSID ""  
#define PASSWORD ""
```

Setup your SSID and Password between the "" in the code.

You can setup your own MQTT server or connect to the OWL DMS (Coming Soon!) here:
5. Upload the firmware

Finally, upload this code to the Duck device by hitting the right-pointing arrow in the top left corner.

This will show a lot of activity in the console, and if successful will say complete. If this process fails for any reason, contact us and we can help debug whatever is going wrong.

Disconnect the USB cable and connect the battery to the board.

If everything worked properly, the OLED screen on the board may light up and you should see the Wifi network available if you check wifi settings on your phone or computer.

Finished Duck:
FAQs

Errors when compiling
1. fatal error: PubSubClient.h: No such file or directory
   - The cause of this error because the library is missing. To install the library, go to Sketch -> Include Library -> Manage Libraries. In the search box, type PubSub. Find and install PubSubClient by Nick O'Leary.

Setup Issues
1. Board not displaying in Port
   a. Make sure that the LoRa board is connected to the computer using a DATA TRANSFER CABLE. Not all cables are created equal. One of the cables transfer data from computer to computer. The other cable just charges devices but cannot transfer/receive data. Make sure you are using a data transfer cable.

Cloud Errors
1. Data not uploading to the internet
   - Double check to see if the credentials (network and password) for WiFi are correctly inputted in the credentials.h file.

API Reference And Quickstart CLusterDuck Protocol
**Quick Start**

Open new sketch in Adruino IDE and include the ClusterDuck library

```cpp
#include "ClusterDuck.h"
```

Create ClusterDuck object

```cpp
ClusterDuck duck;
```

Initializes the ClusterDuck class object

**In setup()**

```cpp
duck.begin(baudRate);
```

Initializes the baud rate for serial printing and messaging. You can adjust to your desired baud rate.

- `int baudRate` -- Default is 115000

Set device ID and captive portal form length.

```cpp
duck.setDeviceId(String deviceId, const int formLength);
```

- `String deviceId` -- input the device ID used to identify your registered device on the web -- do not leave null or empty string
- `const int formLength` -- (optional) define the number of captive portal form fields -- Default is 10 to match our default captive portal template
Setup DuckLink

```cpp
duck.setupDuckLink();
```

`duck.setupMamaDuck` can also be used here to setup a MamaDuck, however you cannot use both in the same sketch.

In loop()

Add corresponding Duck run code. Must be of the same device type as used in `setup()`. (e.g. if `duck.setupMamaDuck()` is used in `setup()` use `duck.runMamaDuck()`)  

```cpp
duck.runDuckLink();
```

Your sketch should look something like this:

```cpp
#include "ClusterDuck.h"
ClusterDuck duck;

void setup() {
  // put your setup code here, to run once:
  duck.begin();
  duck.setDeviceId("Z", 10);
  duck.setupDuckLink();
}

void loop() {
  // put your main code here, to run repeatedly:
  duck.runDuckLink();
}
```

Now compile and upload to your device. If using a Heltec LoRa ESP32 board you should see a Duck Online message on the LED screen. You can now open your phone or laptop’s wifi preferences and connect to the SOS DuckLink Network!

API

```cpp
setDeviceId(String deviceId, const int formLength)
```
- Set device ID and captive portal form length. Do not leave deviceId null or as an empty string. formLength defaults to 10. Use in setup().

```cpp
void begin(int baudRate)
```

- Initialize baud rate for serial. Use in setup().

```cpp
void setupDisplay(String deviceType)
```

- Initializes LED screen on Heltec LoRa ESP32 and configures it to show status, device ID, and the device type. Use in setup().

```cpp
void setupLoRa(long BAND, int SS, int RST, int DI0, int TxPower)
```

- Initializes LoRa radio. If using Heltec LoRa ESP32 set SS to , RST to and DIO to . TxPower corresponds to the the transmit power of the radio (max value: 20). Use in setup().

```cpp
void setupPortal(const char *AP)
```

- Initializes the captive portal code. *AP is the value that will be displayed when accessing the wifi settings of devices such as smartphones and laptops. Use in setup().

```cpp
bool runCaptivePortal()
```

- Processes requests coming through the captive portal. Returns true if there is a new submission. Use this in your loop() function to run continuously.

```cpp
void setupDuckLink()
```

- Template for setting up a DuckLink device. Use in setup().

```cpp
void runDuckLink()
```

- Template for running core functionality of a DuckLink. Use in loop().

```cpp
void setupMamaDuck()
```

- Template for setting up a MamaDuck device. Use in setup().

```cpp
void runMamaDuck()
```

- Template for running core functionality of a MamaDuck. Use in loop().
String * getPortalDataArray()

- Returns webserver arguments based on formLength as an array of Strings.

String getPortalDataString()

- Returns webserver arguments based on formLength as a single String with arguments separated by *

void sendPayloadMessage(String msg)

- Packages msg into a LoRa packet and sends over LoRa. Will automatically set the current device's ID as the sender ID and create a UUID for the message.

void sendPayloadStandard(String msg, String senderId = "", String messageId = "", String path = "")

- Similar to and might replace sendPayloadMessage(). senderId is the ID of the originator of the message. messageId is the UUID of the message. ms is the message payload to be sent. path is the recorded pathway of the message and is used as a check to prevent the device from sending multiple of the same message.

void couple(byte byteCode, String outgoing)

- Writes data to LoRa packet. outgoing is the payload data to be sent. byteCode is paired with the outgoing so it can be used to identify data on an individual level. Reference setDeviceId() for byte codes. In addition it writes the outgoing length to the LoRa packet.
- Use between a LoRa.beginPacket() and LoRa.endPacket() (note: LoRa.endPacket() will send the LoRa packet)

String readMessages(byte mLength)

- Returns a String. Used after LoRa.read() to convert LoRa packet into a String.

bool checkPath(String path)

- Checks if the path contains deviceId. Returns bool.

String * getPacketData(int pSize)

- Called to iterate through received LoRa packet and return data as an array of Strings.
• Note: if using standard byte codes it will store senderId, messageld, payload, and path in a Packet object. This can be accessed using `getLastPacket()`

```cpp
void restartDuck()
```

• If using the ESP32 architecture, calling this function will reboot the device.

```cpp
void reboot(void *)
```

• Used to call `restartDuck()` when using a timer

```cpp
void imAlive(void *)
```

• Used to send a '1' over LoRa on a timer to signify the device is still on and functional.

```java
String duckMac(boolean format)
```

• Returns the MAC address of the device. Using `true` as an argument will return the MAC address formatted using ':'

```java
String uuidCreator()
```

• Returns as String of 8 random characters and numbers

```java
String getDeviceId()
```

• Returns the device ID

```java
Packet getLastPacket()
```

• Returns a Packet object containing senderId, messageld, payload, and path of last packet received.
• Note: values are updated after running `getPacketData()`