```
#include <FileIO.h>
#include <Console.h>
#include <Process.h>
#include <SPI.h>
#include <LoRa.h>
const String Sketch_Ver = "single_pkt_fwd_v004";

static float freq, txfreq;
static int SF, CR, txsf;
static long BW, preLen;
static long old_time = millis();
static long new_time;
static unsigned long newtime;
const long sendpkt_interval = 10000;  // 15 seconds for replay.
const long interval = 60000;          //1min for feeddog (60).
unsigned long previousMillis = millis();
unsigned long previousMillis_1 = millis();

void getRadioConf();//Get LoRa Radio Configure from LG01
void setLoRaRadio();//Set LoRa Radio
void receivepacket();// receive packet
void sendpacket(); //send join accept payload
void emitpacket(); //send ddata down
void writeVersion();
void feeddog();

static uint8_t packet[256];
static uint8_t message[256];
static uint8_t packet1[64];
static int send_mode = 0; /* define mode default receive mode */

//Set Debug = 1 to enable Console Output;
const int debug = 0;
int iuu = 0;
static int packetSize;
static char dwdata[32] = {'\0'};  // for data down payload

void setup() {
  // Setup Bridge
  Bridge.begin(115200);

  // Setup File IO
  FileSystem.begin();
  if ( debug > 0 )
  {
    Console.begin();
    //Print Current Version
    Console.print(F("Sketch Version:"));
    Console.println(Sketch_Ver);
  }


  //write sketch version to Linux
  writeVersion();

  //Get Radio configure
  getRadioConf();

  if ( debug > 0 )
  {
    Console.println(F("Start LoRaWAN Single Channel Gateway"));
    Console.print(F("RX Frequency: "));
    Console.println(freq);
    Console.print(F("TX Frequency: "));
    Console.println(txfreq);
    Console.print(F("Spread Factor: SF"));
    Console.println(SF);
    Console.print(F("TX Spread Factor: SF"));
    Console.println(txsf);
    Console.print(F("Coding Rate: 4/"));
    Console.println(CR);
    Console.print(F("Bandwidth: "));
    Console.println(BW);
    Console.print(F("PreambleLength: "));
    Console.println(preLen);
  }

  if (!LoRa.begin(freq))
    if ( debug > 0 ) Console.println(F("init LoRa failed"));
  setLoRaRadio();// Set LoRa Radio to Semtech Chip
  delay(1000);
  mcu_boot();
}

void loop() {
  if (!send_mode) {
    receivepacket();          /* received message and wait server downstream */
  } else if (send_mode == 1) {
    sendpacket();
  } else {
    emitpacket();
//    sendpacket();
  }

  unsigned long currentMillis = millis();
  if ((currentMillis - previousMillis ) >= interval) {
    previousMillis = currentMillis;
    feeddog();

  }
}

//Get LoRa Radio Configure from LG01
void getRadioConf() {

  char tmp[32];

  Process p;    // Create a process

  //Read frequency from uci ####################
  int j = 0;
  memset(tmp, 0, sizeof(tmp));
  p.begin("uci");
  p.addParameter("get");
  p.addParameter("lorawan.radio.rx_frequency");
  p.run();    // Run the process and wait for its termination
  while (p.available() > 0 && j < 9) {
    tmp[j] = p.read();
    j++;
  }
  freq = atof(tmp);

  //Read txfre from uci ####################
  j = 0;
  memset(tmp, 0, sizeof(tmp));
  p.begin("uci");
  p.addParameter("get");
  p.addParameter("lorawan.radio.tx_frequency");
  p.run();    // Run the process and wait for its termination
  while (p.available() > 0 && j < 10) {
    tmp[j] = p.read();
    j++;
  }
  txfreq = atof(tmp);

  //Read Spread Factor ####################
  j = 0;
  memset(tmp, 0, sizeof(tmp));
  p.begin("uci");
  p.addParameter("get");
  p.addParameter("lorawan.radio.SF");
  p.run();    // Run the process and wait for its termination
  while (p.available() > 0 && j < 3) {
    tmp[j] = p.read();
    j++;
  }

  SF = atoi(tmp) > 0 ? atoi(tmp) : 10;  //default SF10

  //Read tx Spread Factor ####################
  j = 0;
  memset(tmp, 0, sizeof(tmp));
  p.begin("uci");
  p.addParameter("get");
  p.addParameter("lorawan.radio.TXSF");
  p.run();    // Run the process and wait for its termination
  while (p.available() > 0 && j < 3) {
    tmp[j] = p.read();
    j++;
  }

  txsf = atoi(tmp) > 0 ? atoi(tmp) : 10;  //Txsf default to sf9

  //Read Coding Rate  ####################
  j = 0;
  memset(tmp, 0, sizeof(tmp));
  p.begin("uci");
  p.addParameter("get");
  p.addParameter("lorawan.radio.coderate");
  p.run();    // Run the process and wait for its termination
  while (p.available() > 0 && j < 2) {
    tmp[j] = p.read();
    j++;
  }
  CR = atoi(tmp);

  //Read PreambleLength
```

```
  j = 0;
  memset(tmp, 0, sizeof(tmp));
  p.begin("uci");
  p.addParameter("get");
  p.addParameter("lorawan.radio.preamble");
  p.run();    // Run the process and wait for its termination
  while (p.available() > 0 && j < 5) {
    tmp[j] = p.read();
    j++;
  }
  preLen = atol(tmp);

  //Read BandWidth  #######################

  j = 0;
  memset(tmp, 0, sizeof(tmp));
  p.begin("uci");
  p.addParameter("get");
  p.addParameter("lorawan.radio.BW");
  p.run();    // Run the process and wait for its termination
  while (p.available() > 0 && j < 2) {
    tmp[j] = p.read();
    j++;
  }

  switch (atoi(tmp)) {
    case 0: BW = 7.8E3; break;
    case 1: BW = 10.4E3; break;
    case 2: BW = 15.6E3; break;
    case 3: BW = 20.8E3; break;
    case 4: BW = 31.25E3; break;
    case 5: BW = 41.7E3; break;
    case 6: BW = 62.5E3; break;
    case 7: BW = 125E3; break;
    case 8: BW = 250E3; break;
    case 9: BW = 500E3; break;
    default: BW = 125E3; break;
  }
}

void setLoRaRadio() {
  LoRa.setFrequency(freq);
  LoRa.setSpreadingFactor(SF);
  LoRa.setSignalBandwidth(BW);
  LoRa.setCodingRate4(CR);
  LoRa.setSyncWord(0x34);
  LoRa.setPreambleLength(preLen);
}


//Receiver LoRa packets and forward it
void receivepacket() {
  // try to parse packet
  LoRa.setSpreadingFactor(SF);
  LoRa.receive(0);

  unsigned long currentMillis_1 = millis();
  if ((currentMillis_1 - previousMillis_1 ) >= sendpkt_interval ) {
```

```
previousMillis_1 = currentMillis_1;
packetSize = LoRa.parsePacket();

if (packetSize) {   // Received a packet
  if ( debug > 0 ) {
    Console.println();
    Console.print(F("Get Packet:"));
    Console.print(packetSize);
    Console.println(F(" Bytes"));
  }
  // read packet
  int i = 0;

  memset(message, 0, sizeof(message)); /* make sure message is empty */

  while (LoRa.available() && i < 256) {
    message[i] = LoRa.read();



    if ( debug > 0 )  {
      Console.print(F("["));
      Console.print(i);
      Console.print(F("]"));
      Console.print(message[i], HEX);
      Console.print(F(" "));
    }

    i++;

  }    /* end of while lora.available */

  if ( debug > 0 ) Console.println("");

  /*cfgdata file will be save rssi and packetsize*/
  File cfgFile = FileSystem.open("/var/iot/cfgdata", FILE_WRITE);
  cfgFile.print("rssi=");
  cfgFile.println(LoRa.packetRssi());
  cfgFile.print("size=");
  cfgFile.println(packetSize);
  cfgFile.close();

  File dataFile = FileSystem.open("/var/iot/data", FILE_WRITE);
  dataFile.write(message, i);
  dataFile.close();

  if ((int)message[0] == 0) {      /* Join Request */
    send_mode = 1;  /* change the mode */
    return;
  }

  /* process Data down */
  char devaddr[12] = {'\0'};
  sprintf(devaddr, "%x%x%x%x", message[1], message[2], message[3], message[4]);
  if (strlen(devaddr) > 8) {
    for (i = 0; i < strlen(devaddr) - 2; i++) {
      devaddr[i] = devaddr[i + 2];
    }
  }
```

```
      devaddr[i] = '\0';

      memset(dwdata, 0, sizeof(dwdata));
      snprintf(dwdata, sizeof(dwdata), "/var/iot/%s", devaddr);

      if (debug > 0) {
        Console.print(F("Devaddr:"));
        Console.println(dwdata);
      }

      int res = FileSystem.exists(dwdata);
      if (res) {
        send_mode = 2;
        if (debug > 0) {
          Console.print(dwdata);
          Console.println(F(" Exists"));
        }
      }

      Console.print(F("END A PACKET, Mode:"));
      Console.println(send_mode, DEC);
      return; /* exit the receive loop after received data from the node */
    } /* end of if packetsize than 1 */
  } /* end of recive loop */

}

void sendpacket()
{

  int i = 0;

  old_time = millis();
  new_time = old_time;

  while (new_time - old_time < sendpkt_interval) { /* received window may be closed
after 10 seconds */

    new_time = millis();

    if (FileSystem.exists("/var/iot/dldata") == false) {
      delay(1000);
      continue;
    }

    File dlFile = FileSystem.open("/var/iot/dldata"); /* dldata file save the
downstream data */

    memset(packet, 0, sizeof(packet));
    i = 0;
    while (dlFile.available() && i < 256) {
      packet[i] = dlFile.read();
      i++;
    }

    dlFile.close();

    if (i < 3) {
      delay(200);
      continue;
    }

    if ( debug > 0 ) {
      int j;
      Console.println(F("Downlink Message:"));
      for (j = 0; j < i; j++) {
        Console.print(F("["));
        Console.print(j);
        Console.print(F("]"));
        Console.print(packet[j], HEX);
        Console.print(F(" "));
      }
      Console.println();
    }

    new_time = millis();


    while (new_time - old_time < sendpkt_interval - 2000) {   // 8 seconds for
sending packet to node
      LoRa.beginPacket();
      LoRa.write(packet, i);
      LoRa.endPacket();
      delay(1);
      new_time = millis();
    }

    LoRa.setFrequency(txfreq);
    LoRa.setSpreadingFactor(txsf);    /* begin send data to the lora node, lmic use
the second receive window, and SF default to 9 */
    delay(2);

    while (new_time - old_time < sendpkt_interval + 2000) {   // 12 seconds for
sending packet to node

      LoRa.beginPacket();
      LoRa.write(packet, i);
      LoRa.endPacket();
      delay(1);
      new_time = millis();
    }
    LoRa.setFrequency(freq);
    LoRa.setSpreadingFactor(SF);    /* reset SF to receive message */

    if (debug > 0) Console.println(F("[transmit] END"));
    break;
  }

  Process rm;
  rm.begin("rm");
  rm.addParameter("-rf");
  rm.addParameter("/var/iot/dldata");
  rm.run();

  send_mode = 0;
}

void emitpacket()
```

```
{
  int i = 0, j = 0;

  File dwFile = FileSystem.open(dwdata); /* dldata file save the downstream data */

  memset(packet, 0, sizeof(packet));

  while (dwFile.available() && i < 256) {
    packet[i] = dwFile.read();
    i++;
  }

  dwFile.close();

  if (i < 3)
    return;

  if ( debug > 0 ) {

    Console.println(F("Downlink Message:"));
    for (j = 0; j < i; j++) {
      Console.print(F("["));
      Console.print(j);
      Console.print(F("]"));
      Console.print(packet[j], HEX);
      Console.print(F(" "));
    }
    Console.println();
  }

  for (j = 0; j < 5; j++) {     // send data down two times every frequency
    LoRa.beginPacket();
    LoRa.write(packet, i);
    LoRa.endPacket();
    delay(10);

    LoRa.setFrequency(txfreq);
    LoRa.setSpreadingFactor(txsf);    /* begin send data to the lora node, lmic use
the second receive window, and SF default to 9 */
    delay(20);

    LoRa.beginPacket();
    LoRa.write(packet, i);
    LoRa.endPacket();

    delay(20);

    LoRa.setFrequency(freq);
    LoRa.setSpreadingFactor(SF);    /* reset SF to receive message */
    delay(500);
  }

  if (debug > 0) Console.println(F("[transmit] Data Down END"));

  Process rm;
  rm.begin("rm");
  rm.addParameter("-rf");
  rm.addParameter(dwdata);
  rm.run();
  send_mode = 0; //back to receive mode
}

void feeddog() {
  int k = 0;
  memset(packet1, 0, sizeof(packet1));

  Process p;    // Create a process
  p.begin("date");
  p.addParameter("+%s");
  p.run();
  while (p.available() > 0 && k < 32) {
    packet1[k] = p.read();
    k++;
  }
  newtime = atol(packet1);

  File dog = FileSystem.open("/var/iot/dog", FILE_WRITE);
  dog.println(newtime);
  dog.close();

}

//Function to write sketch version number into Linux.
void writeVersion() {
  File fw_version = FileSystem.open("/var/avr/fw_version", FILE_WRITE);
  fw_version.print(Sketch_Ver);
  fw_version.close();
}

void mcu_boot() {
  Process r;
  r.begin("logger");
  r.addParameter("\"mcu_boot\"");
  r.run();
}
```