G62 LoRaWAN Integration

Revision 1.3 - 8 March 2022

1. INTRODUCTION

This document is protocol specification for the Digital Matter G62 LoRaWAN GPS device. Contact info@digitalmatter.com for more information.

1.1. Revision History

Date	Version	Changes
2018-07-06	1.0	Initial Release.
2018-04-16	1.1	Changes to example in section 4.1.
2019-10-03	1.2	Added JavaScript decoder. Corrected accelerometer wakeup time units in downlink 1. Corrected GPS speed accuracy units in downlink 2. Added downlinks 10 and 11.
2022-03-08	1.3	Added timestamps to uplinks 1 and 3, and downlink 1.

2. DEVICE BEHAVIOUR

The G62's behaviour can be configured by setting a variety of parameters using a USB programming adapter, or by sending a subset of those parameters during a downlink message. The details of these parameters can be found on the support website.

2.1. Uplink

Uplinks can occur for multiple reasons, depending on what the devices parameters are set to. These include the following:

- Trip start or end
- Periodically during a trip
- Heartbeat (periodically when not in trip)
- Periodic runtime & odometer updates
- Digital input change
- External power change
- Analog input value change

Due to the limitations of the LoRaWAN protocol (and certain regions), uplinks may be queued for several minutes before being sent on the network. If the device cannot send a message for 15 minutes, the uplink will be cancelled (and the data lost). Care should be taken when setting up a device for a specific region.

2.2. Trips

The device uses the term 'trips' to determine when the device is in use (or not). Trips can currently be started from 3 sources:

Ignition digital input – Started when the ignition input sees a voltage greater than 2v.

Run detect – External input voltage going above, or below, pre-set thresholds.

Movement – The internal accelerometer can detect vibrations from movement, and the trip is started when the device moves a certain distance away from its start position.

The current source of trip is sent with the data message. Trip sources do not “overwrite” other sources; a source needs to end a trip before a different source can start one.

2.3. Odometer & Runtime

During a trip, runtime and odometer counters are constantly being incremented. These values are stored over the life of the device. They are not lost when power is removed.

2.4. Inputs

The G62 LoRaWAN has two digital inputs, and one analog input. This can be used to notify the current state of an external device, or alert when certain inputs change. These can be set up in the configuration app.

2.5. Digital Output

The digital output can be controlled with LoRaWAN downlink messages.

2.6. Downlink

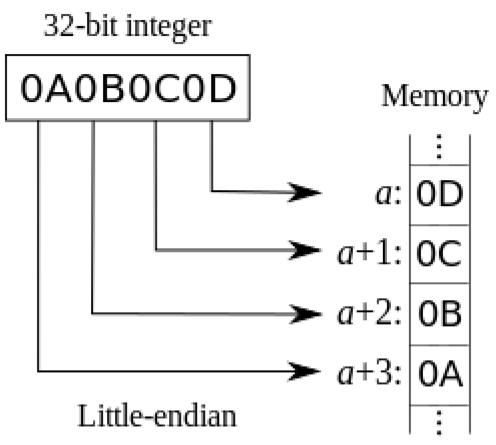
LoRaWAN potentially allows for a downlink with every regular data uplink, but most networks will only allow a few per day. When a parameter update is sent down to the G62, it responds with an application layer acknowledgement uplink as soon as possible. In some regions, this could take several minutes. If the next uplink received is not the expected acknowledgement, the downlink should be resent.

3. NUMBER FORMATS

3.1. Little Endian

Except where noted, all data in the payloads is LITTLE ENDIAN. Be aware of this when converting data fields that consist of more than one byte from the data payload.

https://en.wikipedia.org/wiki/Endianness#Little



3.2. Signed (Negative) Numbers

When a field is specified as *signed*, it is represented in 'two's complement' form. Be aware of this when converting signed fields from the data payload. Where unspecified, assume that fields are unsigned.

https://en.wikipedia.org/wiki/Two's_complement

4. UPLINK MESSAGES

LoRaWAN uplink payload limits can be as small as 11 bytes in some regions (for the longest-range transmissions). The packet headers already include the device serial number, and a 'port number' from 1 to 223, which we will use as a message type.

4.1. Uplink Port 1: Full Data Message

Length = 17 bytes, or 19 bytes including the optional timestamp. If the selected region and parameters don't support sending all bytes at once, this message will be split into 2 separate messages (see Data Message Part 1 & 2).

Offset	Description
0.0-0.1	Trip type: 0 = no trip, 1 = ignition, 2 = GPS or accelerometer movement, 3 = run detect
0.2	External power good flag
0.3	GPS position is current (could get GPS fix)
0.4 (INT28)	28 bit latitude, **signed,** 1 = 0.0000016°
4.0	Ignition state
4.1	Digital input 1 state
4.2	Digital input 2 state
4.3	Digital out state
4.4 (INT28)	28 bit longitude, **signed**, 1 = 0.0000016°
8 (BYTE)	Heading, 1 = 2°
9 (BYTE)	Speed, 1 = 1 km/h
10 (BYTE)	Battery voltage, 1 = 20 mV
11 (UINT16)	External voltage, 1 = 1 mV
13 (UINT16)	Analog input voltage, 1 = 1 mV
15 (INT8)	Internal temperature, 1 = 1 °C
16 (BYTE)	GPS position accuracy, 1 = 1 m
17 (UINT16)	Optional timestamp, supported from firmware v2.4 onwards

The timestamp is not sent by default. It can be enabled in the Tracking parameters, using the configuration tool, or using downlink 1. When present, it records the time that the data record was compiled. So if the GPS position is not current (see offset 0.3), the timestamp will not correspond to the GPS position.

To save space, and therefore allow more frequent uplinks, the timestamp is shortened to only two bytes long. This allows you to resolve the time with a precision of ±1 second, but only if you know the approximate reception time to within ±9 hours.

If the G62 doesn't yet know the current time (ie. has not yet completed a successful GPS fix), the timestamp will be 65535 (0xFFFF), to indicate an invalid time. Otherwise, it is calculated as:

$$Timestamp = UnixTimeSeconds \text{ modulo } 65535$$

Please note that in order to provide a distinct 'invalid' value, the modulo is taken against 65535, not the more typical 65536. The recommended algorithm for recovering the complete time, given an approximate reception time *RxUnixTimeSecs*, and a tolerable total clock drift *MaxTxRxTimeError*, is:

$$RefTime = RxUnixTimeSecs + MaxTxRxTimeError$$
$$RefMult = \text{floor}\left(\frac{RefTime}{65535}\right)$$
$$RefMod = RefTime \text{ modulo } 65535$$
$$TxUnixTimeSecs = \begin{cases} RefMult \times 65535 + Timestamp, & RefMod > Timestamp \\ (RefMult - 1) \times 65535 + Timestamp, & RefMod \leq Timestamp \end{cases}$$

RefTime represents the latest possible time a timestamp could be, accounting for any errors in the G62 and the gateway's clocks. The recommended value of *MaxTxRxTimeError* is 1800 seconds. The timestamp will then be resolved to within a window of [+0.5, -17.5] hours around the approximate reception time, with one second precision.

Example: AA26F5EC16A108450A12CAC9330000171C2318

- A
  - 1010 in binary
  - 10 – Movement trip
  - 0 – No external power
  - 1 – Gps fix is current
- A026F5EC little endian (lowest nibble set to 0)
  - ECF526A0 in hex
  - -319478112 in decimal (signed)
  - -31.9478112° - Latitude
- 6
  - 0110 in binary
  - 0 – Ignition input low
  - 1 – Digital input 1 high
  - 1 – Digital input 2 high
  - 0 – Digital output off
- 10A10845 little endian (lowest nibble set to 0)
  - 4508A110 in hex
  - 1158193424 in decimal (signed)
  - 115.8193424° - Longitude
- 0A
  - 10 x 2 = 20° - Heading

- 12
  - 18 km/h – Speed
- CA
  - 202 x 20 mV = 4040 mV (4.04 V) – Battery voltage
- C933 little endian
  - 33C9 hex
  - 13257 decimal (unsigned)
  - 13.257 V – External voltage
- 0000 little endian
  - 0000 hex
  - 0 decimal (unsigned)
  - 0 V – Analog input
- 17 (signed)
  - 23°C – Internal temperature
- 1C
  - 28 m – GPS position accuracy
- 2318 (optional)
  - Timestamp = 0x1823 = 6179
  - Assume *RxUnixTimeSecs* = 2022-03-04 14:03:00 UTC = 1646402580
  - *RefTime* = 1646402580 + 1800 = 1646404380
  - *RefMod* = 1646404380 modulo 65535 = 34110
  - *RefMult* = floor(1646404380 / 65535) = floor(25122.52) = 25122
  - *RefMod* > *Timestamp,* so:
    TxUnixTimeSecs = *RefMult* x 65535 + *Timestamp*
    = 25122 x 65535 + 6179
    = 1646376449
    = 2022-03-04 06:47:29 UTC

4.2. Uplink Port 2: Data Message Part 1

Length = 11 bytes.

Offset	Description
0.0-0.1	Trip type: 0 = no trip, 1 = ignition, 2 = GPS or accelerometer movement, 3 = run detect
0.2	External power good flag
0.3	GPS position is current (could get GPS fix)
0.4 (INT28)	28 bit latitude, **signed,** 1 = 0.0000016°
4.0	Ignition state
4.1	Digital input 1 state
4.2	Digital input 2 state
4.3	Digital out state
4.4 (INT28)	28 bit longitude, **signed**, 1 = 0.0000016°
8 (BYTE)	Heading, 1 = 2°
9 (BYTE)	Speed, 1 = 1 km/h

10 (BYTE)	Battery voltage, 1 = 20 mV

4.3. Uplink Port 3: Data Message Part 2

Length = 6 bytes, or 8 bytes including the optional timestamp.

Offset	Description
0 (UINT16)	External voltage, 1 = 1 mV
2 (UINT16)	Analog input voltage, 1 = 1 mV
4 (INT8)	Internal temperature, 1 = 1 °C
5 (BYTE)	GPS position accuracy, 1 = 1 m
6 (UINT16)	Optional timestamp, supported from firmware v2.4 onwards

4.4. Uplink Port 4: Odometer & Run Hours Message

Length = 8 bytes.

Offset	Description
0 (UINT32)	Total device runtime, 1 = 1 s
4 (UINT32)	Odometer, 1 = 10 m

4.5. Uplink Port 5: Downlink Ack

Offset	Description
0.0 - 0.6	Sequence number (identifies downlink to server)
0.7	0: Downlink rejected, 1: Downlink accepted
1	Firmware major version
2	Firmware minor version

4.6. Example JavaScript Decoder

```
function Decoder(bytes, port)
{
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};

  if (bytes === null)
    return null;

  if (port === 1)
  {
    if ((bytes.length != 17) && (bytes.length < 19))
      return null;

    decoded._type = "full data";

    switch (bytes[0] & 0x3)
    {
      case 0: decoded.tripType = "None"; break;
      case 1: decoded.tripType = "Ignition"; break;
      case 2: decoded.tripType = "Movement"; break;
      case 3: decoded.tripType = "Run Detect"; break;
    }
    decoded.latitudeDeg = (bytes[0] & 0xF0) + bytes[1] * 256 +
                           bytes[2] * 65536 + bytes[3] * 16777216;
    if (decoded.latitudeDeg >= 0x80000000) // 2^31
      decoded.latitudeDeg -= 0x100000000;  // 2^32
    decoded.latitudeDeg /= 1e7;

    decoded.longitudeDeg = (bytes[4] & 0xF0) + bytes[5] * 256 +
                            bytes[6] * 65536 + bytes[7] * 16777216;
    if (decoded.longitudeDeg >= 0x80000000) // 2^31
      decoded.longitudeDeg -= 0x100000000;  // 2^32
    decoded.longitudeDeg /= 1e7;

    decoded.vExtGood = ((bytes[0] & 0x4) !== 0) ? true : false;
    decoded.gpsCurrent = ((bytes[0] & 0x8) !== 0) ? true : false;

    decoded.ignition = ((bytes[4] & 0x1) !== 0) ? true : false;
    decoded.digIn1   = ((bytes[4] & 0x2) !== 0) ? true : false;
    decoded.digIn2   = ((bytes[4] & 0x4) !== 0) ? true : false;
    decoded.digOut   = ((bytes[4] & 0x8) !== 0) ? true : false;

    decoded.headingDeg = bytes[8] * 2;
    decoded.speedKmph = bytes[9];
    decoded.batV = bytes[10] * 0.02;

    decoded.vExt = 0.001 * (bytes[11] + bytes[12] * 256);
    decoded.vAin = 0.001 * (bytes[13] + bytes[14] * 256);

    decoded.tempC = bytes[15];
    if (decoded.tempC >= 0x80) // 2^7
      decoded.tempC -= 0x100;  // 2^8

    decoded.gpsAccM = bytes[16];

    if (bytes.length < 19)
    {
      decoded.timestamp = null;
      decoded.time = null;
    }
    else
    {
      decoded.timestamp = bytes[17] + bytes[18] * 256;
      decoded.time = ResolveTime(decoded.timestamp, new Date())
      if (decoded.time != null)
        decoded.time = decoded.time.toISOString();
    }

    // Clean up the floats for display
    decoded.latitudeDeg = parseFloat(decoded.latitudeDeg.toFixed(7));
    decoded.longitudeDeg = parseFloat(decoded.longitudeDeg.toFixed(7));
    decoded.batV = parseFloat(decoded.batV.toFixed(3));
    decoded.vExt = parseFloat(decoded.vExt.toFixed(3));
    decoded.vAin = parseFloat(decoded.vAin.toFixed(3));
  }
  else if (port === 2)
  {
    if (bytes.length != 11)
      return null;

    decoded._type = "data part 1";

    switch (bytes[0] & 0x3)
    {
      case 0: decoded.tripType = "None"; break;
      case 1: decoded.tripType = "Ignition"; break;
      case 2: decoded.tripType = "Movement"; break;
      case 3: decoded.tripType = "Run Detect"; break;
    }

    decoded.latitudeDeg = (bytes[0] & 0xF0) + bytes[1] * 256 +
                           bytes[2] * 65536 + bytes[3] * 16777216;
    if (decoded.latitudeDeg >= 0x80000000) // 2^31
```

```
      decoded.latitudeDeg -= 0x100000000;  // 2^32
    decoded.latitudeDeg /= 1e7;

    decoded.longitudeDeg = (bytes[4] & 0xF0) + bytes[5] * 256 +
                           bytes[6] * 65536 + bytes[7] * 16777216;
    if (decoded.longitudeDeg >= 0x80000000) // 2^31
      decoded.longitudeDeg -= 0x100000000;  // 2^32
    decoded.longitudeDeg /= 1e7;

    decoded.vExtGood = ((bytes[0] & 0x4) !== 0) ? true : false;
    decoded.gpsCurrent = ((bytes[0] & 0x8) !== 0) ? true : false;

    decoded.ignition = ((bytes[4] & 0x1) !== 0) ? true : false;
    decoded.digIn1   = ((bytes[4] & 0x2) !== 0) ? true : false;
    decoded.digIn2   = ((bytes[4] & 0x4) !== 0) ? true : false;
    decoded.digOut   = ((bytes[4] & 0x8) !== 0) ? true : false;

    decoded.headingDeg = bytes[8] * 2;
    decoded.speedKmph = bytes[9];
    decoded.batV = bytes[10] * 0.02;

    // Clean up the floats for display
    decoded.latitudeDeg = parseFloat(decoded.latitudeDeg.toFixed(7));
    decoded.longitudeDeg = parseFloat(decoded.longitudeDeg.toFixed(7));
    decoded.batV = parseFloat(decoded.batV.toFixed(3));
  }
  else if (port === 3)
  {
    if ((bytes.length != 6) && (bytes.length < 8))
      return null;

    decoded._type = "data part 2";

    decoded.vExt = 0.001 * (bytes[0] + bytes[1] * 256);
    decoded.vAin = 0.001 * (bytes[2] + bytes[3] * 256);

    decoded.tempC = bytes[4];
    if (decoded.tempC >= 0x80) // 2^7
      decoded.tempC -= 0x100;  // 2^8

    decoded.gpsAccM = bytes[5];

    if (bytes.length < 8)
    {
      decoded.timestamp = null;
      decoded.time = null;
    }
    else
    {
      decoded.timestamp = bytes[6] + bytes[7] * 256;
      decoded.time = ResolveTime(decoded.timestamp, new Date())
      if (decoded.time != null)
        decoded.time = decoded.time.toISOString();
    }

    // Clean up the floats for display
    decoded.vExt = parseFloat(decoded.vExt.toFixed(3));
    decoded.vAin = parseFloat(decoded.vAin.toFixed(3));
  }
  else if (port === 4)
  {
    if (bytes.length != 8)
        return null;

    decoded._type = "odometer";

    var runtimeS = bytes[0] + bytes[1] * 256 + bytes[2] * 65536 + bytes[3] * 16777216;
    decoded.runtime = Math.floor(runtimeS / 86400) + "d" +
                      Math.floor(runtimeS % 86400 / 3600) + "h" +
                      Math.floor(runtimeS % 3600 / 60) + "m" + (runtimeS % 60) + "s";
    decoded.distanceKm = 0.01 * (bytes[4] + bytes[5] * 256 +
                                 bytes[6] * 65536 + bytes[7] * 16777216);
    // Clean up the floats for display
    decoded.distanceKm = parseFloat(decoded.distanceKm.toFixed(2));
  }
  else if (port === 5)
  {
    if (bytes.length != 3)
        return null;

    decoded._type = "downlink ack";

    decoded.sequence = (bytes[0] & 0x7F);
    decoded.accepted = ((bytes[0] & 0x80) !== 0) ? true : false;
    decoded.fwMaj = bytes[1];
    decoded.fwMin = bytes[2];
  }

  return decoded;
}

function ResolveTime(timestamp, approxReceptionTime)
{
    if (timestamp === 65535)
        return null;

    var approxUnixTime = Math.round(approxReceptionTime.getTime() / 1000);

    // Device supplies a unix time, modulo 65535.
    // We're assuming that the packet arrived some time BEFORE refTime,
    // and got delayed by network lag. So we'll resolve the timestamp to
    // somewhere in the 18 hours before the reception time, rather than
    // symetrically in a +- 9 hour window.

    // Wind the reception time forward a bit, to tolerate clock errors.
    var refTime = approxUnixTime + 1800;

    //                          refTime
    //                             v
    // [              |              |              |              ]
    //          ^              ^              ^              ^
    //       timestamp      timestamp      timestamp      timestamp

    //                             refTime
    //                                v
    // [              |              |              |              ]
    //          ^              ^              ^              ^
    //       timestamp      timestamp      timestamp      timestamp

    // We want the timestamp option immediately to the left of refTime.
    var refTimeMultiple = Math.floor(refTime / 65535);
    var refTimeModulo = refTime % 65535;
    var closestUnixTime = 0;

    if (refTimeModulo > timestamp)
        closestUnixTime = refTimeMultiple * 65535 + timestamp;
    else
        closestUnixTime = (refTimeMultiple - 1) * 65535 + timestamp;

    return new Date(closestUnixTime * 1000);
}
```

5. DOWNLINK MESSAGES

LoRaWAN downlink payloads can be as small as 11 bytes in some regions (for the longest-range transmissions). The packet headers already include a ‘port number’ from 1 to 223, which we will use as a message type. The G62 sends an explicit acknowledgement uplink (port 5) on reception of a downlink. It sends the acknowledgement only once, as soon as possible (limited by network, and after any already pending messages). The uplink includes a sequence number to help identify the specific downlink being acknowledged, despite any queuing / buffering in the network. ‘Confirmed’ and ‘unconfirmed’ downlinks are handled in the same way.

5.1. Downlink Port 1: Set Trip Parameters

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Reserved, set to zero
1.0	Enable movement trips, default true
1.1	Enable ignition trips, default true
1.2	Enable run detect trips, default false
1.3	Transmit runtime/odometer at trip end, default false
1.4	Enable timestamps in uplinks 1 and 3, default false, supported from v2.4
1.5 - 1.7	Reserved, set to zero
2 (BYTE)	In trip upload period, 1-127: 1-127 seconds, 129-255: 1-127 minutes, 0 or 128 disables, default 10 mins (138)
3 (BYTE)	Heartbeat period, 10 minutes, 0 disables, default 2 hrs (12)
4 (BYTE)	Odometer/Runtime upload period, 1-255: 1-255 hours, 0 disables, default 1 day (24)
5 (BYTE)	Movement trip speed threshold, 0-255: 0-255 km/h, default 10 km/h
6 (BYTE)	Accelerometer wakeup threshold, 1-8: 63-504 mG, default 63 mG (1)
7 (BYTE)	Accelerometer wakeup time, 1-96: 10-960 ms, 0: 10 ms, default 10 ms (0)

5.2. Downlink Port 2: Set GPS Parameters

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Require 3D GPS fixes, default true
1 (BYTE)	Required PDOP for valid GPS, 25-100: 2.5 to 10.0, default 10.0 (100)
2 (BYTE)	Required position accuracy for valid GPS, 5-100: 5-100 m, default 75 m
3 (BYTE)	Required speed accuracy for valid GPS, 1-2: 10-20 km/h, default 10 km/h (1), finer control available in downlink 11
4 (BYTE)	Discard first N GPS points to allow solution to settle, 0-32: 0-32 positions, default 3
5 (BYTE)	Static hold threshold, 0-90: 0-9.0 km/h, default 5 km/h (50)

5.3. Downlink Port 3: Set Digital Input Parameters

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)

0.7	Reserved, set to zero
1.0	External voltage: Transmit on low to high, default true
1.1	External voltage: Transmit on high to low, default true
1.2	Ignition Input: transmit on low to high, default false (If using ignition-based trips then leave this false)
1.3	Ignition Input: transmit on low to high, default false (If using ignition-based trips then leave this false)
1.4	Digital input 1: Transmit on low to high, default false
1.5	Digital input 1: Transmit on high to low, default false
1.6	Digital input 2: Transmit on low to high, default false
1.7	Digital input 2: Transmit on high to low, default false
2 (BYTE)	Ignition input debounce period, 0-255: 0-2.55 s, default 1 s (100)
3 (BYTE)	Digital input 1 debounce period, 0-255: 0-2.55 s, default 1 s (100)
4 (BYTE)	Digital input 2 debounce period, 0-255: 0-2.55 s, default 1 s (100)

5.4. Downlink Port 4: Set Analog Input Parameters

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Upload on alarm reset, default true
1 (UINT16)	Analog input alarm high threshold, 1-30000: 1-30000 mV, 0 disables, default disabled (0)
3 (UINT16)	Analog input alarm low threshold, 1-30000: 1-30000 mV, 0 disables, default disabled (0)
5 (BYTE)	Hysteresis, 1-255: 10-2550 mV, default 100 mV (10)

5.5. Downlink Port 5: Set Run Detect Parameters

Note: Both thresholds must be valid if run detect is being used.

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Reserved, set to zero
1 (BYTE)	Run detect start time, 0-255: 0-255 s, default 5 s (5)
2 (BYTE)	Run detect end time, 0-255: 0-255 s, default 20 s (20)
3 (UINT16)	Run detect high threshold, 0-36000: 0-36000 mV, default 10 V (10000)
5 (UINT16)	Run detect low threshold, 0-36000: 0-36000 mV, default 5 V (5000)

5.6. Downlink Port 6: Set LoRaWAN Channels

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Reserved, set to zero
1.0 - 1.3	Minimum data rate to use when ADR is disabled, default is 0 (DR0)
1.4 - 1.7	Maximum data rate to use when ADR is disabled, default is 2 (DR2)
2 - 10	Uplink channel mask, set bits are enabled channels, LSb of the 1st byte is channel 0, MSb of the 9th byte is channel 71, set all zeros (default) for the region-specific defaults

The G62 will spread its transmissions out over the allowed data rates in such a way as to equalize the time spent on-air at each data rate. For the default setting of DR0-DR2, this gives a 16 / 30 / 54% split between the three data rates, and maximizes the gateway's capacity. However, the relative range of the three data rates are 100, 75, and 50% respectively. When ADR is enabled, the network server controls the data rate instead.

The uplink channel mask should be left 0 (default) in regions where the network join channels are fixed. In these regions, the gateway will tell the G62 which channels to use, during the join procedure.

In regions where the join channels are not specified (US902-928, AU915-928), you should set the channel mask to avoid continued transmission on unused channels. In these regions the gateway will usually not tell the G62 which channels to use, resulting in significant packet loss if the mask hasn't been programmed.

5.7. Downlink Port 7: Set LoRaWAN Application

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Set AppEUI, 0: Use default AppEUI, 1: Use supplied AppEUI, default 0
1 - 8	AppEUI, **big endian**, ie. default AppEUI 70-B3-D5-70-50-00-00-05 is encoded with first byte as 0x70 and the second byte 0xB3, where 70-B3-D5-70-5 is the manufacturer identifying portion of the AppEUI

The acknowledgement will be transmitted **once** on the existing JoinEUI, and then the Oyster will switch to the new JoinEUI. It continues to use the already provisioned NwkKey and AppKey, which cannot be programmed over the air.

Note that:

- In LoRaWAN 1.0, the JoinEUI is known as the AppEUI
- In LoRaWAN 1.1, changing the JoinEUI resets three cryptographic counters:
    - RJCount1
    - DevNonce
    - JoinNonce
- So after changing the JoinEUI in 1.1, you must reset the counters on the join server

5.8. Downlink Port 8: Set Advanced LoRaWAN Options

Offset	Description

0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Reserved, set to zero
1 (BYTE)	Days between network joins, 0 disables, default 14
2.0 - 2.1	ADR support, 0: Never, 1: When out-of-trip, 2: Always, default never
2.2 - 2.7	Reserved, set to zero

For a detailed description of these parameters, please see the *Configuration and Usage Guide* on the support website.

5.9. Downlink Port 9: Set Digital Output

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Digital output state to be written, 0: off (high impedance), 1: on (switched to ground), factory default is off

Note: The state of the digital output is stored in non-volatile memory, so is not lost on power loss.

5.10. Downlink Port 10: Set Advanced LoRaWAN Options 2

Offset	Description
0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7 - 1.3	Reserved, set to zero
1.4 - 1.7	Initial frame repetitions (NbTrans/Reps), 1-15, default 1
2.0 - 2.3	Initial MaxCount0, sets uplinks between Rejoin0 attempts in LoRaWAN 1.1 OTAA, uplink interval equals 2^(4+MaxCount0), default 15
2.4 - 2.7	Initial MaxTime0, sets approx. time between Rejoin0 attempts in LoRaWAN 1.1 OTAA, uplink period equals 2^(10+MaxTime0) s, default 15
3.0 - 3.3	Initial AdrAckLimitExp, sets uplinks between ADR confirmation requests, uplink interval equals 2^AdrAckLimitExp, default 6 (limit is 64)
3.4 - 3.7	Initial AdrAckDelayExp, sets uplinks between ADR backoff steps, uplink interval equals 2^AdrAckDelayExp, default 5 (delay is 32)
4 (INT8)	Maximum Tx power limit, **signed**, -128-127: -128 to 127 dBm EIRP, default 127 (no limit)
5 (BYTE)	Random Tx delay, 0: disabled, 1-127: 1-127 seconds, 129-255: 1-127 minutes, default disabled

This downlink is supported from firmware version 2.0 onwards. For a detailed description of these parameters, please see the *Configuration and Usage Guide* on the support website.

5.11. Downlink Port 11: Set GPS Parameters 2

Offset	Description

0.0 - 0.6	Downlink sequence number (reported in acknowledgement)
0.7	Require 3D GPS fixes, default true
1 (BYTE)	Required PDOP for valid GPS, 25-100: 2.5 to 10.0, default 10.0 (100)
2 (BYTE)	Required position accuracy for valid GPS, 5-100: 5-100 m, default 75 m
3 (BYTE)	Required speed accuracy for valid GPS, 8-55: 2.88-19.8 km/h, default 9.72 km/h (27)
4 (BYTE)	Discard first N GPS points to allow solution to settle, 0-32: 0-32 positions, default 3
5 (BYTE)	Static hold threshold, 0-90: 0-9.0 km/h, default 5 km/h (50)

This downlink is supported from firmware version 2.0 onwards. It corrects a mistake in the speed accuracy units of downlink 2.

6. CONTACT INFORMATION

For the latest version of this document plus other product information please visit our website at www.digitalmatter.com/support, or contact DM at info@digitalmatter.com.