Mopeka Products, LLC
1223 Industrial St., Suite A, New Braunfels Texas 78130
support@mopeka.com

Mopeka Pro ✓ Sensors Developers Integration Guide

The Mopeka Pro ✓ sensor is a small, battery powered sensor that magnetically mounts to the bottom of a propane tank and uses ultrasound to measure the height of the fluid in the tank. Versions also exist for measuring water and other types of tanks, while being mounted either from the bottom or top of the holding tank.

The sensor uses Bluetooth low energy (BLE) to transmit advertisement packets to a receiver or phone application. This advertisement contains the fluid level, button state, temperature, quality of reading, battery level, sensor id, and the sensors mounting angle

The purpose of this document is to describe the format of the advertisement packet and provide software developers with enough information to be able to integrate the Mopeka Pro ✓ sensors into their own applications. It is expected that the reader already has an understanding of the BLE protocol and the ability to scan for and read advertisements on their chosen platform.

Version History

Version	Date	Author	Description
1	09/22/2019	JLK	Initial version
2	06/12/2020	JLK	Updates for level as mm and other errors
3	10/1/2020	JLK	Reverted to support firmware sending raw level and added the conversion/compensation function to perform scaling to height in mm. The hwid values were updated to maintain backward compatibility for firmware v0.0.74. However, this mode and version has been deprecated since only a controlled number of sensors ever received this firmware.
4	12/2/2020	JLK	Added coefficients for Ammonia (NH3) in comments of the sample code to convert from raw values to height in mm
5	3/23/2021	JLK	Fixed mistake in 0 vs 1-based start bit numbering
6	01/11/2022	JLK	Added coefficients for water (for Pro H20) in comments of sample code
7	09/17/2022	JLK	Updates for PRO+ and TD-40 sensors
8	01/09/23	JLK	Fixed sign on 2nd coefficient for water in sample code comments
9	03/28/23	JLK	Added coefficients for air (for top-down sensors)
10	07/12/23	JLK	Added Univerval Sensor HWID
11	09/25/23	JLK	Corrected extension bit explanation. Added more gas/diesel coefficients, updated text/description of hardware ID or rather product names.

Mopeka Products, LLC
1223 Industrial St., Suite A, New Braunfels Texas 78130
support@mopeka.com

Syncing to Mopeka Pro ✓ Sensors

The Mopeka Pro ✓ sensors do not perform classic BLE "pairing" to send most of their data and instead use a connection-less design where the necessary sensor readings are sent in the Bluetooth advertisement packet. Thus, a BLE scanner will see all Mopeka Pro ✓ sensors in the nearby vicinity, including their sensor readings.

In order to "SYNC" with a specific sensor, it is the application's responsibility to listen only for the MAC address of the specific BLE sensor the customer is using. The sensors transmit the state of the "SYNC" button in their BLE advertisement, and it is the responsibility of the application to provide a special scanning mode where it looks specifically for a user's sensor while the button is pressed. When seen, the application should remember the MAC address of that user's sensor so that the application can filter and display future packets from the respective sensor.

NOTE: The MAC address is sent in every BLE packet as part of the specification. But since Apple iOS does not provide any mechanism to access the MAC address, the Mopeka Pro ✓ sensor also duplicates the last 3-bytes of the MAC address in the BLE advertisement payload data. This can be seen as a unique sensor ID and is suitable to use to identify the SYNC'ed sensors.

Advertisement Payload

The Mopeka Pro ✓ sensor uses the BLE "Manufacturer Specific" advertisement identifier to send its data as follows

Name	Offset	Length	Default Value	Description
Packet Length	0	1	13	Per BLE specificion - length of manufacturer specific data
BLE Flag	1	1	0xFF	Per BLE specification - Identifies manufacturer specific data follows
Manufacturer ID	2	2	0x0059	Per BLE specification - Identifies microcontroller manufacturer. Transmitted least significant byte first
Hardware ID and Extended range bit	4	1		**Bit 0-6:** This ID represents the specific type of PRO sensor as follows: 0x03 = Pro Check, Bottom-up propane sensor 0x04 = Pro-200, top down BLE sensor, wired power 0x05 = Pro Check H20, bottom-up water sensor 0x08 = PRO+ Bottom-up Boosted BLE sensor, all commodities 0x09 = PRO+ Bottom-up BLE+Cellular Sensor, all commodities 0x0A = Top-down Boosted BLE sensor 0x0B = Top-down BLE+Cellular Sensor 0x0C = Pro Universal Sensor, bottom up, all commodities Other = Reserved for OEM or future applications, or documented in other integration guides

				Bit 7: Raw Tank Level extension bit - this bit will only be set when time of flight extends past 16384us. It is more than just a 15th extension bit, but rather changes the resolution to 4us with a 16384us offset, when set. Thus full scale range is 0 to 16383us with 1us resolution like all prior sensors, and then when this bit is set, 16384 to 81920us can be represented with 4us resolution. See sample code below for conversion
Battery Voltage	5	1		**Bit 7:** Reserved **Bits 0-6:** Battery voltage scaled such that [0 to 127] represents value [0 to 3.96875 volts]. In other words, read the lower 7 bits into an integer, and divide by 32 to get the battery voltage
Temperature and button	6	1		**Bit 7:** Sync button state **Bit 0-6:** Temperature, in units of C, with an offset of -40. Full range is thus -40 to 87C in 1C increments .
Raw Tank Level and quality	7	2		Sent least significant byte first **Bits 14-15:** Sensor measurement quality "stars" from 0 to 3. This is an arbitrary number that represents that quality or confidence level of the ultrasonic reading and how it is mounted on the tank **Bits 0-13:** Raw tank level time-of-flight in microseconds, see sample function 'get_tank_height_lpg' below for scaling information.
UID/MAC address	9	3		Least 3 significant bytes of MAC address. Should always be validated that is matches a sensor the user has specifically "Synced" to. These 3 bytes represent the sensor ID.
Sensor mount angle	12	2		**Byte 10:** X acceleration (signed integer from -128 to 127) **Byte 11:** Y acceleration (signed integer from -128 to 127) These two bytes are the accelerometer reading which indicates the angle the sensor is currently mounted in relation to the horizon. The readings measure the gravity vector current applied to the sensor in the range of +/- 0.125G. If held beyond that angle, the reading will saturate or clip to the extremes in a similar manner as a "bubble level" might hit the ends. To convert to G, take the signed integer and divide by 1024.
UUID Service Length	14	1	3	Per BLE Specification - Length of the following UUID service length descriptor
UUID Service Flag	15	1	2	Per BLE Specification - Indicates a 16-bit service UUID packet follows
UUID Service	16	2	0xFEE5	Per BLE Specification - Reported service UUID. On certain phones this can be used to allow filtering for Mopeka Pro ✓ packets and should be used when possible to prevent falsely identifying other BLE devices as a Mopeka Pro ✓ sensor. The least significant byte is sent first.

Receiving Advertisements from Mopeka Pro ✓ Sensors

In order to detect and filter Mopeka Pro ✓ sensor packets, the following pseudo-code is recommended as an example. This example is listening for a previously SYNCed sensor. If the SYNCing process is being done, then the application should instead wait for the appropriate sensor with the SYNC button pressed. It should then save that sensor's MAC address so it can listen for its readings in the future.

```
bool parse_packet(bd_addr addr, byte[] scanData)
{
    // Check for expected length
```

```
    if (scanData.Length != 18) return false;

    // Manufacturer specific
    if (scanData[1] != 0xFF) return false;

    // Length of trailing packet
    if (scanData[0] != 13) return false;

    // Check data in address matches mac - part of our protocol for iOS fix
    if (scanData[9] != addr.Address[2]) return false;
    if (scanData[10] != addr.Address[1]) return false;
    if (scanData[11] != addr.Address[0]) return false;

    // Check for manufacturer code
    if (scanData[2] != 0x59 || scanData[3] != 0x00) return false;

    // Read sensor version - set as appropriate
    prod_version = scanData[4] & 0x7F;
    if (prod_version != 0x03 && prod_version != 0x05) return false;

    // Check that this is our "sync"ed sensor
    if (addr != synced_mac) return false;

    return true;
}
```

Read Sync Button State

In order to sync a Mopeka Pro ✓ sensor to a phone or application, the typical method is to require the user to press the SYNC button. When the SYNC button is pressed, the application can then remember the MAC address so that it can display that sensors readings moving forward.

```
bool is_sync_pressed(uint8_t *scanData)
{
    if (scanData[6] & 0x80) {
        return true;
    }
    return false;
}
```

Manufacturing Mode

All Mopeka Pro ✓ sensors that use CR2032 batteries are shipped in "manufacturing mode" which is used to save battery power. In manufacturing mode, the sensor is asleep and will not transmit any packets. The user must press the SYNC button 5 times in a row to wake the sensor up into its normal operating mode. Each press must come within 60 seconds of the first.

Before all 5 presses are received, the sensor will advertise a packet in "manufacturing mode" as indicated by a 0x0060 "Manufacturer ID" value. This will occur for approximately 60 seconds after each press before the advertising stops again.

The official Mopeka Pro ✓ app does not currently display any information about sensors in manufacturing mode, but an app could display a help tip if it sees this condition. For example, if a sensor in manufacturing mode is detected, the application could display a help prompt saying, "You must press the SYNC button 5 times in a row to wake up your sensor".

NOTE: Please do not rely on any runtime variables when the device is in manufacturing mode. The values and even the presence of manufacturing mode itself are not guaranteed to exist and could be removed without notice in future firmware revisions.

Read Angle of Sensor

The Mopeka Pro ✓ supports accelerometer readings to help assist the end-user with leveling the sensor on the bottom of their propane cylinder or other tank so that it is completely flat with no angle to the horizon.

Below is a code snippet that shows how to decode an X or Y acceleration. This returns a value in units of G's or the magnitude of the gravity vector that the sensor is currently reading. This can then be scaled to an angle relative to the horizon and mapped to a "bubble level" graphic or any other mechanism desired

```
getAcceloY(rawByte) {
    // Return value in "G's" which can be scaled to an angle
    return (char)scanData[12] / 1024;
}

getAcceloX(rawByte) {
    // Return value in "G's" which can be scaled to an angle
    return (char)scanData[13] / 1024;
}
```

Read Measured Tank Height

The following code is sample 'C' code that returns the measured sensor height, in millimeters, assuming that the commodity is liquid propane and a bottom-up sensor. The coefficients in the code sample can be changed for reading water or other liquid commodities. For top-down sensors, only air can be measured so the coefficients for air should be used. This function performs the conversion and compensation based on the raw values transmitted in the advertisement. Contact Mopeka for the conversion, temperature compensation, and coefficients for other fluids/gases – e.g. butane, LPG/butane mixtures, oil, etc.

Note that certain sensors such as the Pro Universal or Pro+ can be used on multiple different liquid commodities and are not targeted only at LPG. For these sensors, the app must request the commodity type, or somehow obtain this from the user, and then use the appropriate coefficients in the code below.

```
// Returns the measured fluid height (or air height), in mm
// scanData - represents the array of raw bytes for the manufacturing data
uint32_t get_tank_height(const uint8_t* scanData)
{
    // For LPG (Liquid propane) use the following coefficients
    const double coef[3] = { 0.573045, -0.002822, -0.00000535 };

    // For NH3 (Ammonia) use the following coefficients instead
    // const double coef[3] = { 0.906410, -0.003398, -0.00000299 };

    // For H20 (water) use the following coefficients instead
    // const double coef[3] = { 0.600592, 0.003124, -0.00001368 };

    // For Top-down sensors (measuring air), use the following coefficients:
    // const double coef[3] = { 0.153096, 0.000327, -0.000000294 };

    // For diesel and gas, use the following coefficients:
    // const double coef[3] = {0.7373417462, -0.001978229885, 0.00000202162};

    // Retrieve raw level from data.  Only the LS 14-bits represent the level
    uint16_t raw = (scanData[8] * 256) + scanData[7];
    double raw_level = raw & 0x3FFF;

    // Check for presence of extension bit on certain hardware/firmware -
    // it will always be 0 on old firmware/hardware and raw value saturates
    // at 16383.  When extension bit is set, the resolution changes to 4us
    // with 16384us offet for wider range.  Thus legacy sensors and
    // firmware still have 0 to 16383us range with 1us, and new versions
```

```
    // add the range 16384us to 81916us with 4us resolution
    if (scanData[4] & 0x80) {
        raw_level = 16384.0 + raw_level * 4.0;
    }

    // Retrieve unscaled temperature from advert packet
    double raw_t = (scanData[6] & 0x7F);  // MSbit is button state - ignored

    // Apply 2nd order polynomial to compensating the raw TOF into mm of LPG
    return (uint32_t)(raw_level * (coef[0] + coef[1] * raw_t + coef[2] * raw_t *
raw_t));
}
```

Read Mopeka Pro ✓ Battery Voltage

The following C code shows how the conversion to voltage is performed, as well as how the official Mopeka Pro ✓ App currently converts to battery percentage for CR2032 based sensors.

```
float get_battery_voltage(uint8_t *scanData)
{
     // Return battery voltage – note that MSbit is reserved and should be set 0
     return (float)(scanData[5] & 0x7F) / 32.0f;
}

// Arbitrary scaling of battery voltage to percent for CR2032
float get_battery_percentage(float voltage)
{
    float percent = (voltage - 2.2f) / 0.65f * 100.0f;
    if (percent < 0.0f) {
        return 0.0f;
}
    if (percent > 100.0f) {
        return 100.0f;
    }
    return percent;
}
```

Sensor Temperature Reading

The Mopeka Pro ✓ sensor will provide its microcontroller's on-die temperature sensor reading over the BLE advertisement. This temperature is internally used for temperature compensation and the accuracy is specified by the chip manufacturer with a worst-case accuracy of +/- 5C. No characterization has been done to try to map the on-die temperature to ambient temperature. Its purpose is to provide temperature-based adjustments to both the fluid tank level as well as coin-cell voltage

```
int get_temperature_reading(uint8_t *scanData)
{
    uint8_t temp = scanData[6] & 0x7F;
    return ((int)temp - 40);
}
```

Hyper Mode

Anytime the user presses the SYNC button, the sensor will enter a hyper mode of operation for the next 20 to 30 minutes. When in hyper mode, the sensor will advertise approximately 3 times a second.

This allows for a good user experience when initially SYNCing their sensor to the application. It will greatly improve responsiveness and also will result in the sensor locking onto a good tank level reading much quicker. There is no specific indication of the hyper mode in the data payload. If the application wants to detect this, it can check for any detected advertisement delta shorter than 3 seconds, since this will never happen otherwise