Open Cosmos

www.open-cosmos.com

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| *qbcan CanSat Releaser* User Manual |

CHANGE RECORD

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Issue 1 | | Revision 0 | | | | | | | |
| Reason for change | | Date | | Pages | | Section | | Paragraph(s) | |
|  |  |  |  |  |  |  |  |  |

Table of contents:

[1 Introduction 4](#_Toc446404056)

[1.1 Scope of the Document 4](#_Toc446404057)

[1.2 Applicable and Reference documents 4](#_Toc446404058)

[1.2.1 Applicable Documents (ADs) 4](#_Toc446404059)

[1.2.2 Reference Documents (RDs) 4](#_Toc446404060)

[1.3 Open Cosmos 4](#_Toc446404061)

[1.4 Terminology / Definitions 4](#_Toc446404062)

[2 System Description 5](#_Toc446404063)

[2.1 Microcontroller 5](#_Toc446404064)

[2.2 Transceiver 6](#_Toc446404065)

[2.3 Temperature and Pressure Control 7](#_Toc446404066)

[2.4 Power 7](#_Toc446404067)

[2.5 Releaser Mechanism 7](#_Toc446404068)

[2.6 Library 8](#_Toc446404069)

[3 Software 9](#_Toc446404070)

[3.1 Software Installation 9](#_Toc446404071)

[3.2 Install the qbcan CanSat Releaser examples 9](#_Toc446404072)

[4 Library and Examples 10](#_Toc446404073)

[4.1 CanSat Releaser example 10](#_Toc446404074)

[4.2 Ground Station Releaser example 12](#_Toc446404075)

# Introduction

## Scope of the Document

This document is the qbcan CanSat Releaser user manual and describes the qbcan Releaser hardware and software. This document is based on the *qbcan User Manual* document [AD01].

## Applicable and Reference documents

### Applicable Documents (ADs)

1. Open Cosmos, *qbcan User Manual*, Iss. 1, Rev. 0

<http://doc.open-cosmos.com/qbcan_user_manual>

### Reference Documents (RDs)

## Open Cosmos

Open Cosmos uses nano-satellites to provide simple and affordable access to space to organisations ranging from SMEs and research institutions to space agencies in developing countries.

Open Cosmos is based in London with an international mindset and the ambitious purpose to bring the possibility to use and develop space technology to a wider public. With that in mind, Open Cosmos has developed qbcan, an educational CanSat kit for primary, secondary ant tertiary education. qbcan aims at engaging and motivating the next generation of space engineers and scientists.

## Terminology / Definitions

BMP180 Pressure and temperature sensor.

RFM69 433 MHz transceiver

# System Description

qbcan CanSat Releaser is a releaser system designed to be carried by by a rotor-blade drone vehicle. It is based on the qbcan CanSat bus (see [AD01]) and the CanSat container.

qbcan is a versatile, easy-to-use and high performance CanSat bus. The qbcan bus provides all the required capabilities of a minimalistic CanSat: radio communications, a temperature and pressure sensor and a computing platform with a wide range of interfaces. The user can then add extra functionality (adding sensors and actuators via the provided interfaces) and develop more complex systems.

The main components of qbcan are:

* Arduino Pro Micro microcontroller.
* RFM69HW 433MHz transceiver.
* BMP180 temperature and pressure sensor.
* Rapid-development software library that interfaces with the transceiver and the pressure and temperature sensor.
* 9V battery.
* Two M2 mounting holes.

qbcan (hardware and software) is open-source and users are encouraged to modify it and re-distribute their work.

The CanSat container is composed of a cardboard tube structure, meant to host the CanSat during flight, and the release mechanism and electronics inside another tube in order to be protected.

A the bottom of the tube, there is a hinged door to insert and expel the Cansat.

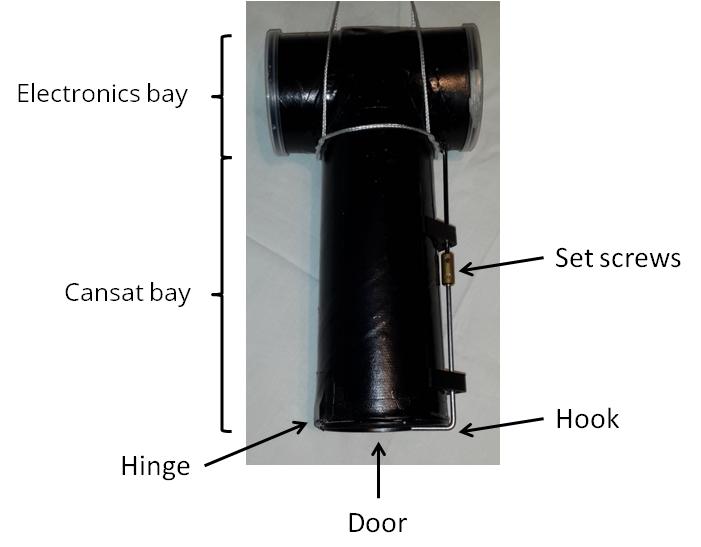


Figure 2‑1: Cansat releaser parts

## Microcontroller

The core of qbcan is an Arduino Pro Micro microcontroller. The microcontroller provides the required computing power to the CanSat. The big number of interfaces provided by the Arduino Pro Micro allows the user to develop more complex CanSat missions by integrating additional peripherals. The main features of the Arduino Pro Micro are:

* ATmega32U4 running at 5V/16MHz.
* Easy to program using the Arduino Integrated development environment.
* On-Board micro-USB connector for programming.
* I2C, SPI and UART serial communication ports.
* 4 channels to read analogue signals using a 10-bit analogue to digital converter.
* 5 Pulse Width Modulated output pins.
* 12 Digital Input Output pins.
* Tiny footprint: 33.0 x 17.8 mm.

The pinout of the Arduino Pro Micro can be seen in Figure 1.

All these pins and communication interfaces are available to the user except pin 7, 10, 11, 12 and 13 which are used to communicate with the transceiver. The I2C interfaces are shared with the pressure and temperature sensors, but are also available to the user.

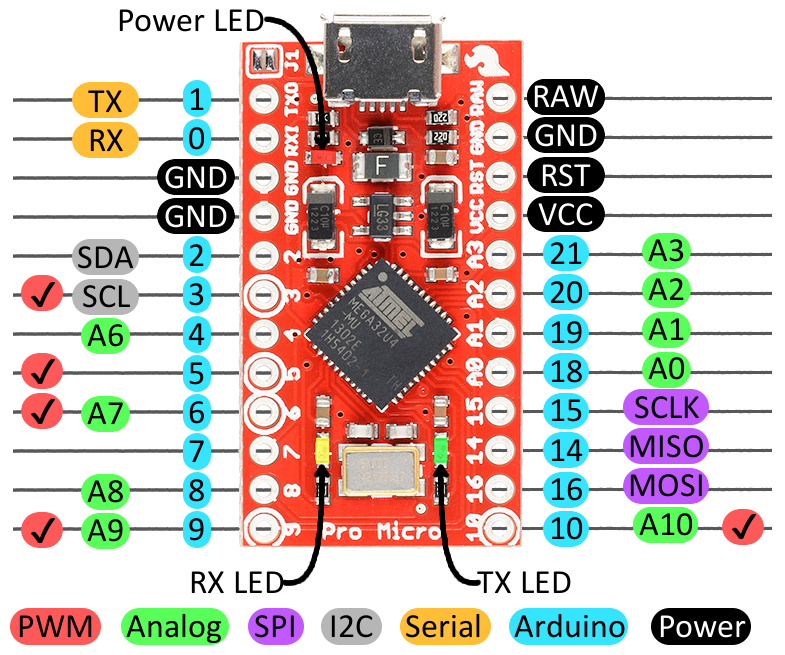


Figure ‑: Arduino Pro Micro pinout

## Transceiver

A RFM69HW 433 MHz transceiver is included to provide long range communication capabilities to qbcan. The main features of the transceiver are:

* +20 dBm - 100 mW power output capability.
* High sensitivity: down to -120 dBm at 1.2 kbps.
* Programmable output power: -18 to +20 dBm in 1 dB steps.
* Fully integrated synthesiser with a resolution of 61 Hz.
* Frequency selectable by software over 256 different channels.
* 255 possible nodes in every channel.
* FSK, GFSK, MSK, GMSK and OOK modulations.
* Hardware 128 bit AES encryption.
* Over 400+ meters range using whip antennas and several km range using a Yagi antenna on the receiving end.

The transceiver communicates with the Arduino over the SPI interface and uses the pin 10 of the Arduino as the Slave Select (that is why this pin is reserved). The transceiver software, included with qbcan, is interrupt driven (asynchronous response to incoming communications) and uses the Arduino pin 7 to provide this interrupt. Hence pin 7 is not available to the user.

The antenna of the transceiver is a simple quarter wavelength monopole antenna.

Using this transceiver, a qbcan can be used in a CanSat while another qbcan is used as ground station, receiving telemetry from the CanSat (and sending to a PC via the USB port) and sending commands to the CanSat.

## Temperature and Pressure Control

qbcan includes a BMP180 barometric pressure and temperature sensor. This sensor communicates over I2C and provides:

* Pressure sensing range: 300-1100 hPa (9000 m to -500 m above sea level).
* Up to 0.02 hPa / 0.17 m altitude resolution.
* -40 to +85 °C operational range, +-2 °C temperature accuracy.

## Power

The qbcan Releaser is powered by a 9 V battery. The connector to the battery is included and by using the available Arduino pins the user is able to access the raw 9 V from the battery - from the Arduino Raw pin - and 5V regulated power - from the Arduino VCC pin status open.

If you plan to use more than 500 mA from the 5V line it is recommended to use the raw battery voltage and use your own voltage regulator.

A companion board containing a transistor circuit is included to the CanSat container electronics. This board draws current directly from the 9 V battery and is used to power the electric motor.

## Releaser Mechanism

The releaser mechanism consists on a motor connected to a shaft that runs parallel to the releaser tube.

At the bottom of the tube, a hinged door is present with the shaft on the opposite side of the hinge. This shaft ends up in a 90 degrees hook which the door rests onto. When the shaft rotates, the hook also rotates and the weight of the Cansat pushes the door open and drops by gravity.

Midway in the shaft, there is a brass fitting with 2 set screws in order to reset the releaser.

## Library

A qbcan Releaser software library is included. It provides an easy-to-use interface with the transceiver and pressure and temperature sensors. An example code is provided to speed up development of the CanSat mission. The library includes a CanSat Releaser example and a Ground Station Releaser example.

# Releaser instructions

## Tools needed

* Hex (Allen) key size 1.5

## Cansat releaser arming instructions

* Make sure the releaser electronics contain the releaser software.
* Make sure releaser electronics are off. The releaser needs to be reset before operation.

To switch it off:

* + Open the cap on the electronics bay
  + Either toggle the power switch if present or disconnect the battery
* Unscrew one of the set screws in the shaft so the hook at the end of the shaft can freely rotate.
* Insert the Cansat into the tube.
* Close the door.
* Rotate manually the hook to the armed position, see below.
* Tighten the set screw with the hex key.
* Switch on the releaser electronics by connecting the battery or toggling the power switch.
* Close the electronics’ cap.

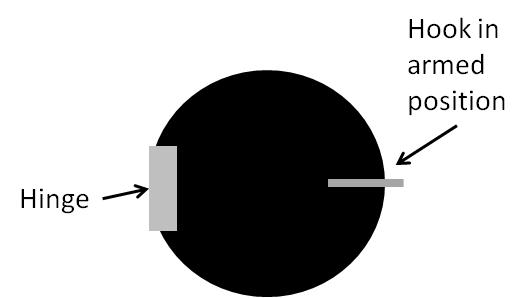


Figure 3‑1: Cansat diagram showing hook in armed position

## Instructions to release the Cansat

* Once the Cansat has been inserted and the releaser has been set, the drone can be flown and once the Cansat needs to be released, the key word can be sent via the serial monitor on the Arduino IDE.

# Software

## Software Installation

The Arduino Pro Micro microcontroller is the core component of the qbcan. Refer to [AD01] for a detailed guide on how to install the Arduino IDE, Arduino Pro Micro drivers and the Arduino qbcan library.

## Install the qbcan CanSat Releaser examples

Once the PC is ready for using the qbcan, proceed to install the qbcan CanSat Releaser examples into the qbcan library folder. To do so, copy the folders CansatReleaser and GroundStationReleaser to the ./Arduino/libraries/qbcan/examples folder in the Arduino sketchbook directory. See [AD01] for more details.

To make sure that the library has been correctly installed, re-launch the Arduino IDE software and check that the qbcan CanSat Releaser examples can be found in File > Examples.

# Library and Examples

The software library used in the qbcan CanSat Releaser is the same as in the qbcan CanSat bus. Refer to [AD01] for a detailed description of the library and its functionalities.

In order to use the library you will need to include the qbcan library and the SPI.h and Wire.h libraries at the beginning of your code, so that the qbcan can have access to the SPI and I2C buses.

//Include the required libraries

#include <qbcan.h>

#include <Wire.h>

#include <SPI.h>

The following sections describe the examples provided with the qbcan Releaser. These examples are modifications of the examples provided with the qbcan CanSat bus, therefore only the remarkable differences are described below.

## CanSat Releaser example

The CanSat Releaser parameters are defined before the setup function.

//Releaser options

int motorTime = 150; // number of miliseconds the motor is ON

int gatePin = 6; // Connected to transistor gate

char releaseTriggerChar = 'R'; // Character used in the command

The motorTime variable defines the time during which the motor that releases the CanSat is switched on. For the sake of simplicity no closed-loop control of the angle of rotation has been implemented.

The gatePin variable defines the Arduino pin used to activate the transistor gate of the companion board that drives the motor.

The releaseTriggerChar variable defines the character that needs to be sent by the Ground Station Releaser qbcan in order to release the CanSat.

The loop function is composed of two parts, namely: the measurement of the temperature and pressure (as featured by the qbcan CanSat bus) and the listening for commands.

When a packet is received from the Ground Station, the message is compared with the release command in the condition radio.DATA[0] == releaseTriggerChar. If the condition is satisfied, the command is triggered.

The release command is executed by the releaseCansat function which sets HIGH and LOW during a defined time (motorTime) the pin (gatePin) that drives the motor.

If acknowledgment is required by the Ground Station command, the qbcan CanSat Releaser acknowledges the reception and execution of the command and sends the current measured temperature and pressure, together with the number of received Release commands.

The radio frequency parameters definition for the CanSat Releaser example is described in [AD01] and [AD02].

void loop()

{

double T,P;

// Get a new pressure reading:

bmp.getData(T,P);

if (radio.receiveDone())

{

if (radio.DATALEN >= 1){

if (radio.DATA[0] == releaseTriggerChar){

// If release cmd is received, call release function

releaseCansat(motorTime, countRelease);

ledOK(true); // LEDs

Serial.print("#[");

Serial.print(++packetCount);

Serial.print(']');

Serial.print('[');Serial.print(radio.SENDERID, DEC);Serial.print("] ");

Serial.print("[Release CMD # ");Serial.print(countRelease);Serial.print("] ");

Serial.print(" [RX\_RSSI:");Serial.print(radio.RSSI);Serial.print("]");

}

} else {

// ADD FUNCTIONALITIES FOR DIFFERENT RECEIVED COMMANDS - HERE

for (byte i = 0; i < radio.DATALEN; i++)

Serial.print((char)radio.DATA[i]);

}

// Send ACK and DATA if requested by the Ground Station

if (radio.ACKRequested())

{

byte theNodeID = radio.SENDERID;

radio.sendACK();

Serial.print(" - ACK sent.");

sprintf(payload,"Released! [#%d] T: %d C, P: %d mb.",(int)countRelease,(int)T,(int)P);

radio.send(GATEWAYID, payload, payloadLEN);

Serial.println(payload);Serial.print(" - Send complete");

}

Serial.println();

}

}

## Ground Station Releaser example

Similarly, the Ground Station Releaser code requires its parameters to be defined accordingly to the former. The releaseTriggerChar must agree with the Cansat Releaser's one.

//Releaser options

char releaseTriggerChar = 'R'; // Character used by the Releaser

bool reqACK = true; // request acknowledgement from Cansat releaser

void loop()

{

// If Serial communication available (any commands from PC)

if (Serial.available() ) {

// User command acquisition

cmdToCansat = charParse(1);

// Send command to cansat

sprintf(payloadToCansat,"%s",cmdToCansat);

radio.send(GATEWAYID, payloadToCansat, 50, reqACK); // with ACK request

Serial.print(payloadToCansat);

Serial.println(" - Send complete");

delay(50);

}

// Listen for transmissions

if (radio.receiveDone())

{

Serial.print("#[");

Serial.print(++packetCount);

Serial.print(']');

Serial.print('[');Serial.print(radio.SENDERID, DEC);Serial.print("] ");

if (promiscuousModeGS)

{

Serial.print("to [");Serial.print(radio.TARGETID, DEC);Serial.print("] ");

}

for (byte i = 0; i < radio.DATALEN; i++)

Serial.print((char)radio.DATA[i]);

Serial.print(" [RX\_RSSI:");Serial.print(radio.RSSI);Serial.print("]");

// Acknowledge if requested

if (radio.ACKRequested())

{

byte theNodeID = radio.SENDERID;

radio.sendACK();

Serial.print(" - ACK sent.");

}

Serial.println();

}

}

char \* charParse (byte length)// Return parsed byte from serial

{

char BUFFER[length + 1]; // Define BUFFER variable

delay(50); // Needed for Baud 9600, less delay for faster speed

// Read Serial and store in BUFFER

for(byte i = 0; i < length; i++) {

BUFFER[i] = Serial.read();// Read serial char by char

}

BUFFER[length] = 0; // Set last value as 0

//return atoi(BUFFER);// Convert string to integer and return

return BUFFER;

}