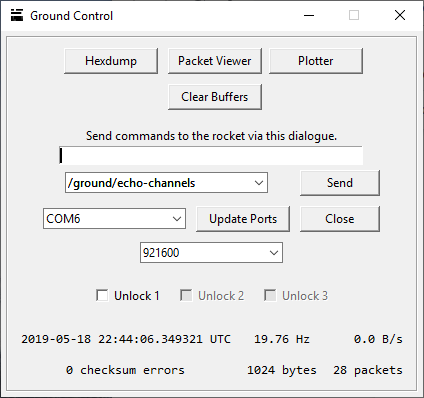
R@VT Ground Controller User Manual

v2019.05.20

**Installation and Setup**

1. Begin with the Beaglebone powered off.
2. Plug the Beaglebone into the Arduino radio node via the USB port. (This Arduino MUST be operating at a serial baudrate of 921600 bps, but can be any kind of Arduino.)
3. Connect the ground station computer to a radio Arduino node via USB. This Arduino can operate at any baudrate.
4. On the ground station computer, open the rvtgui.pyw executable. (Double click this in Windows File Explorer or enter python rvtgui.pyw in a console.)
5. The window below will appear. The port name for the radio Arduino will appear in the ports list. (If not, or if new ports are added, click “Update Ports.”) Click the “Open” button to connect to the radio.

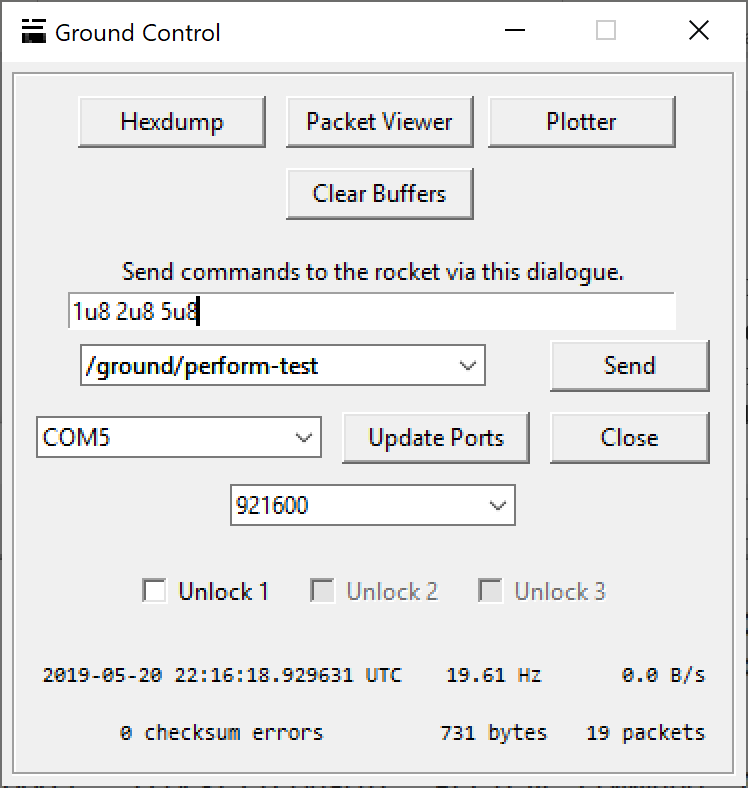


*The ground controller main terminal.*

The main terminal window is the primary interface for the user to communicate with the vehicle. The simplest action is to send an empty packet over a particular channel, which can be selected in the channel dropdown menu. The image shown has the channel “/ground/echo-channels” channel selected. Clicking “Send” in the current configuration would command the rocket to list out all of the channels it has in its database; the output of this command is shown in the following images. Non-empty packets can be sent by entering a formatted string into the text entry box. This string is used to construct packets of arbitrary composition for transmission to the rocket, and warrants its own discussion.

To explain how to send data to the rocket, consider the following use case. The user wants to determine which unit tests are available to the vehicle, and then would like to perform some of these tests. The first step is to send an empty packet to the /ground/echo-tests channel; in response, the rocket will confirm that an “ECHO TESTS” command was received, and then print out all of its supported unit tests. After reviewing the available tests, the user decides they want to perform tests 1, 2, and 5. The PERFORM TESTS command expects an array of unsigned 8-bit integers as an argument, which represent the IDs of tests to perform. To send a command which will achieve this behavior, the user should set the channel to /ground/perform-tests and enter the following string into the text entry box, before hitting send:

1u8 2u8 5u8



This will send a packet with data 0x01 0x02 0x05, three unsigned 8-bit integers. Other formats are possible, such as unsigned and signed integers of other sizes, floats, doubles, and strings. Below is a list of formats, their explanations, and some examples cases, along with examples of message composition.

|  |  |  |  |
| --- | --- | --- | --- |
| Formatter String | Datatype | Size (bytes) | Examples |
| u8 | unsigned char | 1 | 6u8, 159u8 |
| u16 | unsigned short | 2 | 678u16, 3480u16 |
| u32 | unsigned word | 4 | 561u32, 981500u32 |
| u64 | unsigned long | 8 | 85u64, 250000000u64 |
| n8 | signed char | 1 | 78n8, -45n8 |
| n16 | signed short | 2 | 9321n16, -490n16 |
| n32 | signed word | 4 | 80054n32, -7410n32 |
| n64 | signed long | 8 | 73891u64, -283746u64 |
| f | float | 4 | 65.3f, -279.04f |
| d | double | 8 | 65.3d, -279.04d |
| # | string | Variable | #Hello, #“Rockets are cool!” |

A 64-bit unsigned integer, two floats, and a double:

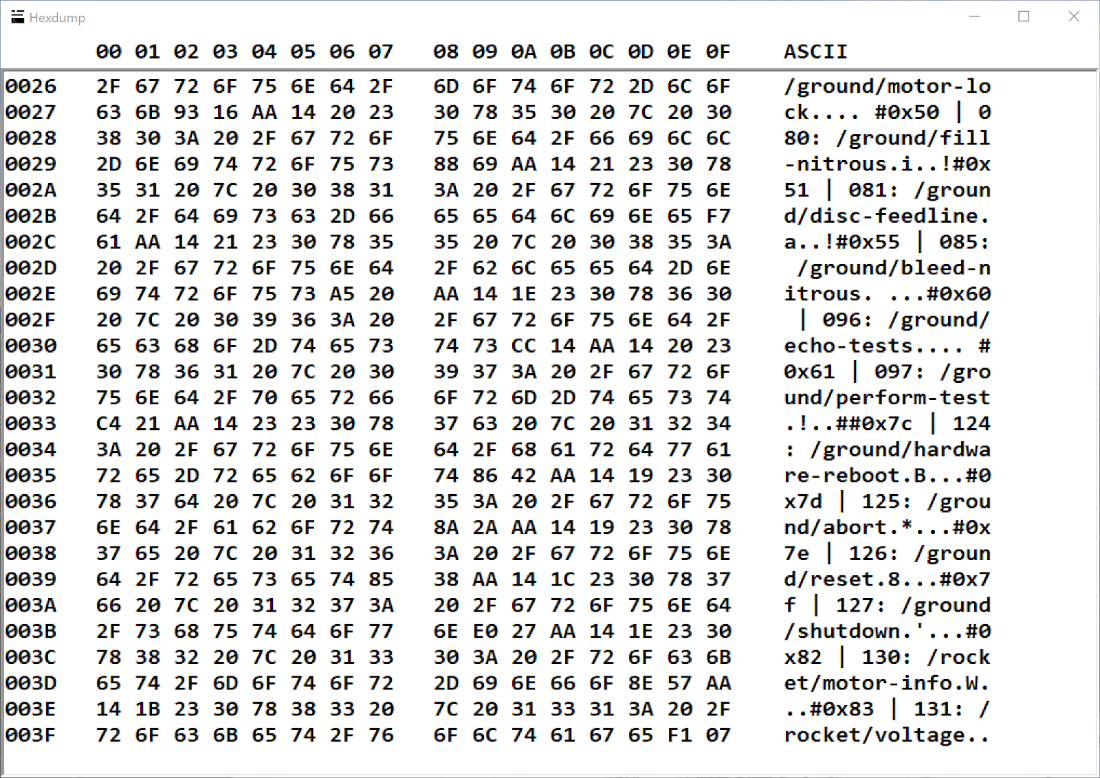
842987220u64 56.43f 89.20f 78.002d

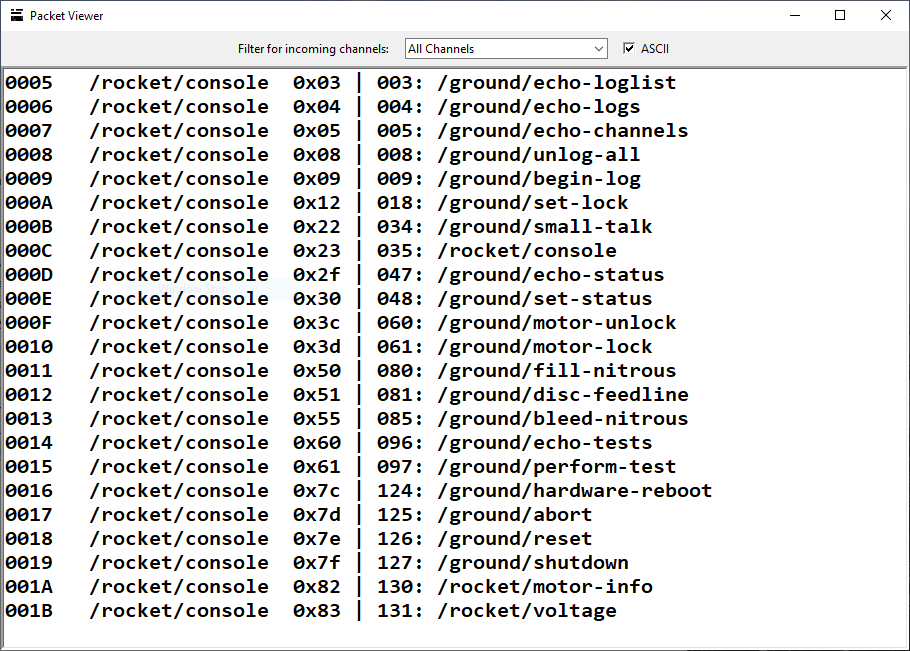
Two strings, and 3 signed 16-bit integers:

#OneWordString #“Multi word string” -563n16 901n16 2733n16

Note that strings containing spaces must be wrapped in quotes to be parsed properly. The quotes should not contain the # character.

Below the generic communication interface are three toggle switches for unlocking motor access. The motor requires three distinct unlock codes, in a unique order, to be armed, and the interface restricts the user to this order. The user should verify that the rocket’s response is as expected when attempting to unlock the motor.





*The Hexdump (above) and Packet Viewer (below) windows.*

To view raw incoming data, click the “Hexdump” button. This opens the Hexdump view, a live updating display of all bytes received, unfiltered, from other radios on the same frequency. Clicking the “Packet Viewer” button will open the Packet Viewer window, which lists all the packets successfully received from the controller in descending order of the time received. Toggling the “ASCII” switch will display the packets as either hexadecimal bytes or ASCII characters; the latter is most useful for reading the console channel, while the former can be used to debug. The Packet Viewer permits users to filter by a particular channel; most human-readable output is sent from the vehicle on the /rocket/console channel. Other logs usually have their own dedicated channel, such as /rocket/voltage.

**Troubleshooting**

Several things could prevent the radios from establishing a proper connection between the motor controller and ground station. The first thing to check is the baudrates; all devices should be operating on a standard baudrate of 921600 bps. One can verify that the connection between the ground station and its radio node is correct by opening the Hexdump window, and opening the Arduino port. Something to the effect of “Lora radio init on 434 Hz” will be received if there are no issues with the ground station configuration. The Beaglebone configuration is much trickier to debug, but great effort has been made so that the common folk won’t have to. As long as the Beaglebone is plugged into a radio Arduino (Uno or Mega) operating at 921600 bps, hardware setup should go off without a hitch.

Potential issues with the operation of this system range in frequency between expected and extremely rare. The most common of these is checksum errors, which result in packet drops; this is indicated by the “checksum errors” counter on the ground control main terminal window. These are rather common. When one of these is encountered, it’s usually not a big deal, but the previous command may have to be retransmitted. How to respond to a dropped packet is left to the user’s discretion; there are no automatic corrections.

Another possible problem during controller operation is that the controller flow of execution will become trapped in an infinite loop, and hang. Many hours of debugging have been put towards preventing this from happening, but it might still crop up. The only way to solve this issue is to reboot the Beaglebone. It’s easy to detect that a program hang has occurred; every iteration through its control loop, the Beaglebone proceeds through an animation on its build-in LEDs. If these LEDs stop animating for an extended period of time, a hang has occurred, and the device should be rebooted. Simply unplug, and restore, power to the device.