Take control of your MINDSTORMS bricks (2)

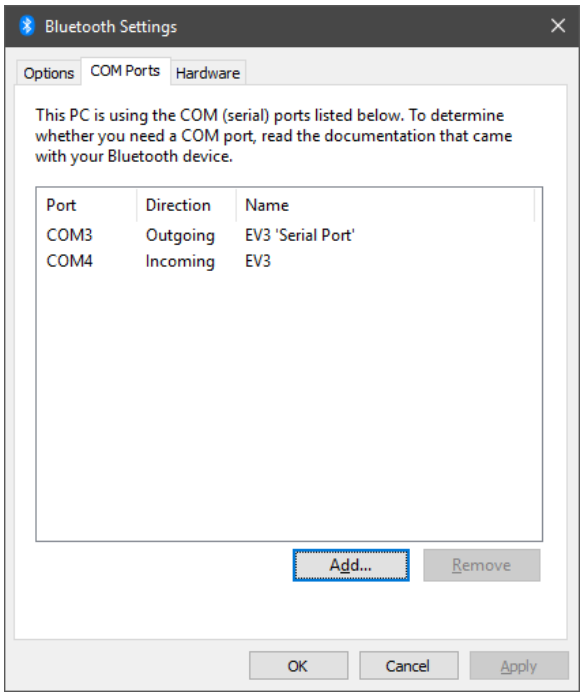
By Oton Ribić

In the wake of the introductory article (HBM 28, page 69) of this series, where we went through the general idea of communicating with our pBricks, let us now get deeper into the matter itself and try to construct our first messages. For now, let's begin with the EV3 pBrick, since its workflow is slightly simpler, and cover the same path for the NXT later. Also, keep in mind that all these instructions work with the factory, stock firmware.

Connection first

The first step in 'talking' to the EV3 pBrick is establishing a connection with it. This is most conveniently done using Bluetooth. The device intended to control the EV3, presumably a computer, needs to be paired with the EV3, which is a fairly straightforward operation done via the device's Pairing option in the Settings. However, before starting to send the messages, a terminal or port needs to be set up, which will be used to communicate with the newly paired pBrick.

On a PC, navigate to Bluetooth Settings, then head to the tab COM Ports, click on Add, and create an Outgoing port to the pBrick you've previously paired with your computer. Note the number your computer will assign to the newly created COM. Without the need to go into deeper technical detail, let's just mention that this is an emulation, or rather a virtual communication port, and is in itself actually based on a rather old (as old as Classic Space), yet reliable and ubiquitous COM standard. All communication to and from MINDSTORMS bricks occurs through such COM ports channeled through the paired Bluetooth connection. In case you were wondering, this is no hack but a completely standard and recommended way of inter-device communication.



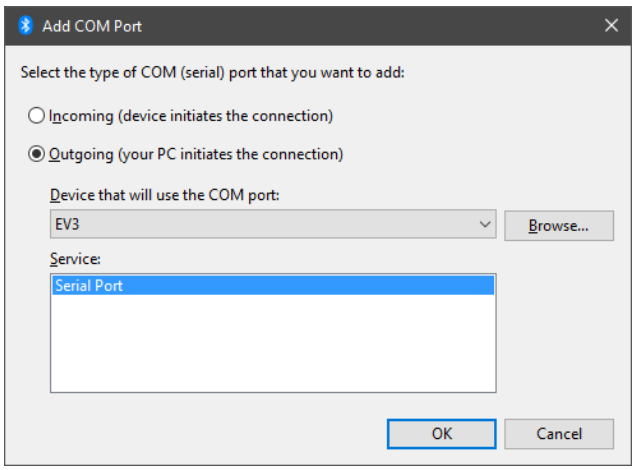
To set up a Bluetooth COM port, head to Bluetooth Settings and click Add

Messaging and what it does

Now that the devices have been paired and their connection channel established, let's look deeper into the structure of the message. Generally, each message (in either direction) is a sequence of bits (i.e. digital zeroes and ones) grouped in sets of eight called — as you probably already know — bytes. Each message must be constructed according to a set of exact rules in order to be received and interpreted correctly. We will get to these rules in a minute, but before that let's take a look at how these messages actually work.

There are exceptions, but in general we can consider each message as a container of an instruction block, or a set of instruction blocks, from the official MINDSTORMS programming bundle, in sequence. For example, a block which lets motor B rotate 600 degrees clockwise at 75% power can be encapsulated into a Bluetooth message for the EV3, along with all these parameters. As we will see in the later articles, this is where EV3 differs from the NXT — and is therefore more practical.

Of course, when controlling a pBrick communication actually happens in both directions: we will usually desire to receive an acknowledgment from the pBrick that the previously sent message has been received and interpreted correctly. Also, in other cases the computer (or any controlling device) may want to read a value from a certain sensor connected to the pBrick; it is then down to the brick to send both an acknowledgment and the read value, possible.



Then create an outgoing port towards your EV3 brick. Note the COM number which will be assigned to it!

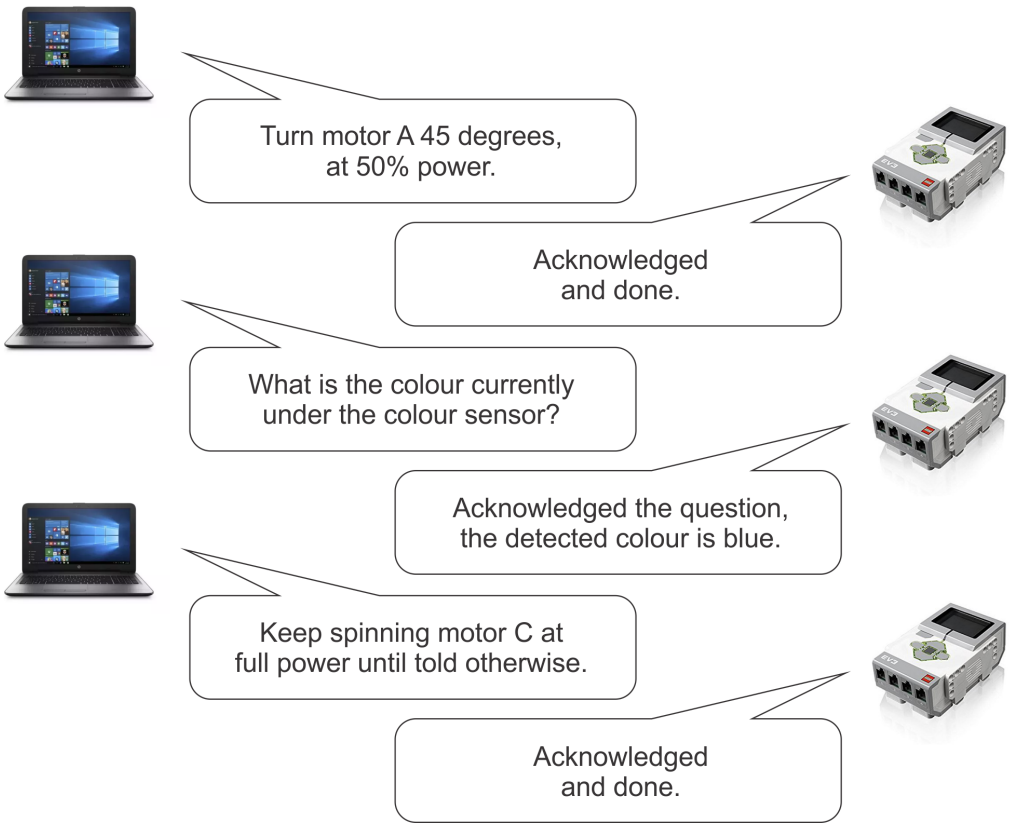
if

This exchange of messages between the two devices can be very rapid in practice, and nearly always begins with the computer sending a message, the pBrick acknowledging and acting upon the instruction, then sending back the success or error result. Whenever we need to ensure each instruction completely finishes before the next one begins, this final message will be a useful indicator that the EV3 has done all it was told to do, and is ready for new instructions.

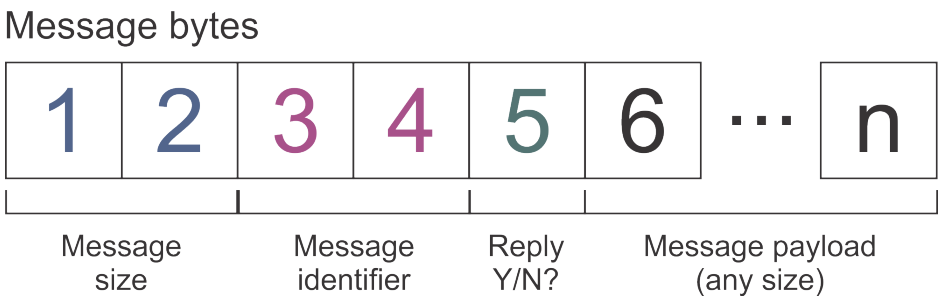
Into bits and bytes

So, what do the messages look like? Let us begin with a few parameters at the top which are common and mandatory for all messages, while the remainder (which is more flexible) will be a subject of later upcoming articles.

Any message, regardless of who sends and who receives, begins by stating its own length, as a sixteen-bit number (from 0 to 65535, split into two bytes with values from 0 to 255) and, according to IT rules, with the smaller byte first and larger second, and not including this very length designator. For example, if the message is 300 characters long, the first length byte to be sent will be 44, and the second 1, because 1×256 + 44 = 300. If you are not at least superficially familiar with how binary and decimal number systems work, now is the time to learn! Wikipedia and many other free internet sites offer a lot of information on these topics.



It may seem redundant to state in advance how long a message is (you don't announce the number of letters in an email you send, do you?), but this is good engineering practice to ensure there are no ambiguities or lost information.
After the first two bytes specifying the length, another 16-bit (two byte) number follows, which is the identifier of the message. It can be any arbitrary number, as it serves only to let the EV3 refer, when responding, to a specific message with that very number. For most cases this is not needed, but in those rare situations (which we won't be covering in this series, but you are welcome to explore for yourself) when multiple instructions and actions are being performed simultaneously, it can be useful to know to which of several previously sent messages the EV3 is referring to when acknowledging or returning a value, etc.



Then, the fifth byte in the message is an indicator for whether an acknowledgment reply from the EV3 pBrick is desired. If zero, the message will require acknowledgment; and if 128, EV3 will just 'silently' accept it. In usual circumstances, one would desire to have every instruction acknowledged, but since this process may delay things for a tenth of second or two, in some very fast and critical movements this may be omitted.

If this is still rather confusing, don't worry. It will all become clearer when we start constructing some example messages in accordance with these rules, in the next article. So stay tuned!
#