

# NXT 001



## An Introduction to Robotics

### Lesson 4

By Chad Cardwell

#### Objectives

- Discuss advanced NXT-G programming blocks and functions
  - Data hubs and wires
  - Data blocks
    - Logic block
    - Math block
    - Compare block
    - Range block
    - Random block
  - Advanced blocks
    - Text block
    - Number To Text block
  - My Blocks
  - Variable block

## Data Hubs and Wires

So far, we've only discussed programming blocks that were self-contained, meaning they were only configured by the values we set in their configuration panels. However, you may remember in Lesson 3 where data wires were briefly mentioned with regards to *Loop* and *Switch* blocks. The *Loop* block has the option of enabling a counter that displays a data plug when enabled. This serves as a numerical output that can then be used elsewhere in the program using a data wire.

Figure 4.1 gives us an overview of how data wires are used and represented. Data wires essentially transfer data from one data plug to another. Data plugs are found on blocks on their data hubs. If you click the lower left portion of the block, its data hub will extend from the bottom of the block similar to what is shown in Figure 4.1. There will usually be input and output plugs on this data hub for each of the block's customizable settings that appear in its configuration panel. It's useful to know that if a data wire is used as an input to a block, any hardcoded or typed configuration data for that input will be overwritten by the data wire.

Data wires can carry three different types of data, which are number, text, and logic. These are color coded as yellow, orange, and green, respectively. To place a data wire, simply click once on an output plug and once again on an input plug of the same data type. The wire will automatically route itself so it can be seen. If a data wire is broken or if the plugs it is connected to don't have the same data type (data mismatch), it will appear as a dotted gray wire. Hovering over it will reveal in the NXT-G software's lower right help menu what the problem is.

With data hubs, data is transferred straight across an input plug to its adjacent output plug. This allows you to place an input data wire into a plug such as the wire labeled [E] is configured in Fig. 4.1. A new data wire can then be extended from the adjacent output plug and used elsewhere in the program. Data wires can be deleted by clicking on them and pressing the Delete key.

## Data Blocks

Located in the complete palette, the *Data* blocks are designed to handle data in a variety of useful ways.

### Logic Block

This block is designed to perform a logical operation on its inputs and output a true/false logical value. Its operations consist of *And*, *Or*, *XOr*, and *Not*. Its inputs can be hardcoded in the configuration panel, but if data wires are used, any hardcoded values will be overwritten.

### Math Block

This highly useful block can perform such math operations on two numbers as *Add*, *Subtract*, *Multiply*, and *Divide*, and it outputs the result as a number value. Like the *Logic* block, its A and B values can either be manually set in the configuration panel or dynamically set with input data wires.

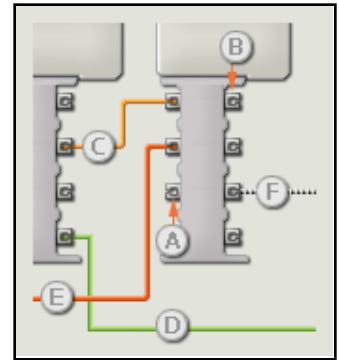


Figure 4.1 – Data hubs and wires (as shown in help tool)

- [A] Input plug
- [B] Output plug
- [C] Number data wire (yellow)
- [D] Logic data wire (green)
- [E] Text data wire (orange)
- [F] Broken data wire (gray)

## Compare Block

This block simply compares two numbers as to whether or not the first number is *greater than*, *less than*, or *equal to* the second number. It outputs a logic value depending on the result of the comparison.

## Range Block

This block, unlike the previously described ones, can take up to three inputs. It checks as to whether or not a test value number is within or outside of a given numerical range. It outputs a logic value depending on the result of the comparison.

## Random Block

Used to output a random number, this block takes two inputs in the form of the min and max values of the random numbers it will be allowed to generate. This block is useful for creating a robot that exhibits unpredictable behavior because the block's output will vary each time the program runs.

## Advanced Blocks

In addition to the *Data* blocks, there is a listing of *Advanced* blocks that allow for even more advanced operations. We will only cover two of these blocks, but they are all listed and explained in the help tool.

## Text Block

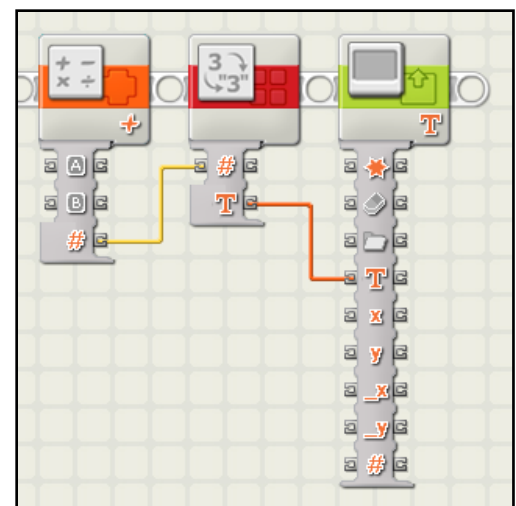
This block allows you to combine up to three different text strings into a single text string. The text strings can be typed or received as inputs through a data wire. The block outputs the combination of the supplied text strings as a single text string.

## Number To Text Block

This block is used to convert a numerical value to its textual character value. It simply takes a single number input and outputs its converted text version. A useful application for this is when you need to display numbers to the NXT Brick screen using a *Display* block. Figure 4.2 shows an example of how this can be accomplished. It uses a *Math* block with user-typed in numbers and outputs their sum. This output is used as an input into the *Number To Text* block, and its text version is output into the *Display* block's text input plug. This example shows how multiple blocks can easily be linked together in the NXT-G software environment.

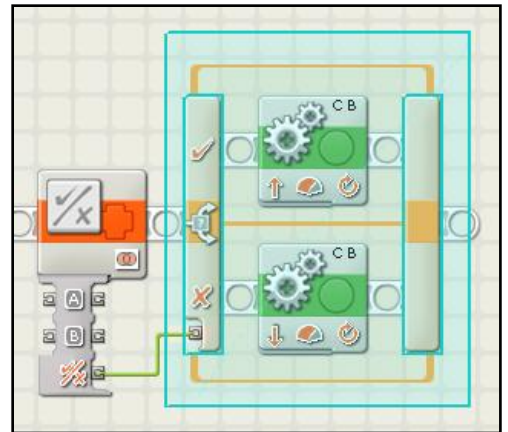
## My Blocks

Sometimes you will create such a useful section of programming code that you will want to reuse it in either multiple parts of a single program or in a completely different program altogether. Rather than have to copy and paste these blocks all over the place, you can create what is known as a *My Block*, which is a block that is representative of a series of other blocks.



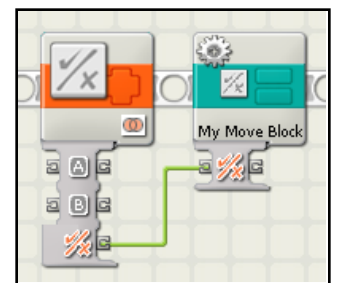
**Figure 4.2 – Number To Text Block in use**

To create a new *My Block*, you must first have a set of programming blocks on the workspace. Simply highlight the blocks you want to make into a *My Block* by clicking and dragging a selection box around them, and then select the *Create My Block* icon at the top of the NXT-G software next to the cursor comment button. This will open the *My Block Builder*, which will display the selected blocks as well as allow you to customize what the *My Block's* icon will look like. First type in a name representative of what the selected blocks do, and type a description if you want for future understanding. After clicking *Next*, you will be presented with an icon builder screen. From here you can drag and drop which icons you want to appear on your *My Block* for visual clues before saving it.



**Figure 4.3 – Creating My Block with input plug and wire**

After a *My Block* has been created, it is saved in the custom palette for future use. While it is represented as a single icon, it actually contains within it all of the blocks you selected when creating it. To see these blocks or to make changes to the *My Block*, double click it and it will open its contents up in a new tab.



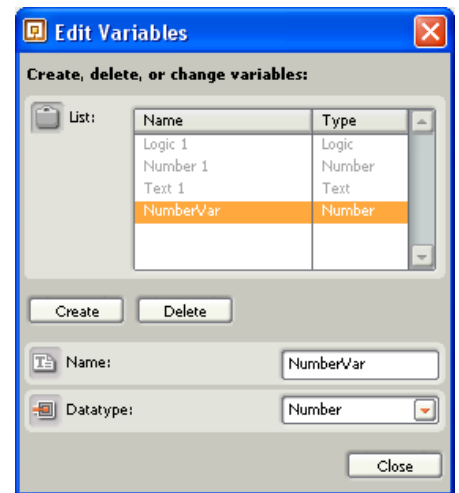
**Figure 4.4 – My Block with input plug and wire**

To create a *My Block* that can input outside values or output a value, meaning that it has a data hub, you have to select an area of blocks that has a data wire entering or exiting the selection area, as shown in Figure 4.3. This will result in the creation of a *My Block* as shown in Figure 4.4, where a data hub is available.

## Variable Block

This block is used for storing data values in the NXT Brick's internal memory. Other blocks can then read or write to the *Variable* block as needed anywhere within the program. To use variables, you must first use the *Define Variable* command in the *Edit* menu. From here you can create new or modify existing variables.

As you can see in Figure 4.5, I have input the information to create a new variable entitled *NumberVar* with the *Number* datatype. I can now configure a *Variable* block to either read or write to the *NumberVar* variable that I defined.



**Figure 4.5 – Define Variables window interface**

Variables are defined per program, so in order to use one in both a program file and a *My Block*, you will have to define it in both files with the same name and include a *Variable* block for the defined variable in both files as well.