RCX Command Center

version 3.1
Written by Mark Overmars

The RCX Command Center was written to more easily work with the Lego MindStorms and CyberMaster robot system. It is built around NQC (Not Quite C Compiler), written by Dave Baum, that makes it possible to program the RCX (the programmable brick that is the heart of the Lego robots) in a language that is close to C. For more information on NQC, see the tutorial and NQC reference guide provided.

The basis of the RCX Command Center consists of an editor in which you can write your NQC programs with possibilities to compile the programs and download them to the RCX. In case of errors they are reported at the bottom of the editor window such that they can be corrected easily. Besides the editor, the RCX Command Center contains tools to control your robot directly, to watch what is happening in the RCX, to download (new) firmware, etc.

To run RCX Command Center it is necessary that spirit.ocx (the ActiveX control for the RCX) is registered in your system. This happens automatically when installing the Lego MindStorms or CyberMaster software. If you don't want to install the Lego software (or can't, e.g. on NT 4) copy the spirit.ocx file from the Lego CD-ROM to you hard disk and use the **Run** command in the windows **Start** menu to execute

```
REGSVR32.EXE spirit.ocx
```

This will register the OCX on your machine. (You might need to specify the path of spirit.ocx in the above command.)

The RCX Command Center can be used free of charge. Please note that NQC and it documentation are copyrighted by Dave Baum. This version of RCX Command Center uses version 2.0 of NQC. (This version is NOT compatible with version 1.x. You can though set the program in compatibility mode in the Preference menu. There is also a command in the Edit menu to translate an old style program into a new style program.) See the readme file for more information on copyright and distribution issues. Please send your comments to markov+lego@cs.uu.nl. See the web page

http://www.cs.uu.nl/~markov/lego/

for more information.

Not Quite C

Not Quite C (NQC) is an easy to use language for programming the Lego robots. The preprocessor and control structures of NQC are very similar to C. NQC is not a general purpose language - there are many restrictions that originate from actual limitations of the RCX.

NQC was written by Dave Baum. The RCX Command Center simply calls the NQC compiler for compiling and downloading code to the RCX. For more information on NQC look in the tutorial (which contains a reference guide) provided with the RCX Command Center. For more information of NQC, see the web page

http://www.enteract.com/~dbaum/lego/nqc/

Starting RCX Command Center

When you start the RCX Command Center, it first asks you for the COM port to connect to the robot. Press **Ok** to automatically detect it or choose the port yourself. (If you press **Cancel**, no connection is made and certain options are not available.) The program now tries to locate the tower used for communication with the RCX. If it cannot find it, certain options are unavailable. Next it tries to find the RCX itself. If it cannot find it you are notified. Switch the RCX on or move the robot closer to the tower, and press **Ok** to try again.

You can also choose between MindStorms and CyberMaster. Please note that for the CyberMaster, once the software has established the connection to the robot, you should not switch the robot off and on again. The CyberMaster requires some special startup sequence whenever you switch it off. If this happens by incident, Choose **Turn RCX Off** from the **Tools** menu, switch the robot back on, and then choose **Find RCX**.

If your tower is always connected to the same port you probably don't want to see the form each time you start the program. This can be changed in the **Preferences** item in the **Edit** menu.

You can also start the program by dragging an NQC file to the icon, or by double clicking on an NQC file and indicating that it should be started with RcxCC.

Editor windows

After starting RcxCC, the main window is shown with a menu bar, a tool bar, a large area in which editor windows can appear, and a status bar. The default way of working is to indicate that you want a new file (in the toolbar or in the **File** menu) or to open an existing file, to write the NQC program in the editor window that appears, and then to download it to the RCX using the **Download (and Run)** command in the **Compiler** menu. If there are errors in your program, a panel appears at the bottom of the editor window indicating the errors. You can click on the errors to make the cursor move to the corresponding place in your code, such that you can correct the errors.

Note that RCX Command Center uses color-coding to make your NQC program more readable. Green indicates comment, maroon indicates define statements, and blue indicates RCX commands and constants. You can switch color-coding off in the preferences. To this end click on **Preferences** in the **Edit** window. Choose the tab **Editor** and click on the appropriate checkmark.

You can open multiple files and work on them simultaneously. Use the commands in the **Window** menu to arrange the different editor windows.

Loading and Saving Files

All file handling is done using the **File** menu. Here you find the usual commands to create a new, empty file, to open an existing file, to save a file, to print it, and to exit the program. You can also use the buttons on the toolbar for most of these commands. You can also open a file by dragging it from your windows explorer to the RcxCC window.

Other commands in the **File** menu include **Insert**, that inserts the contents of a file at the current cursor position, and commands to close the current or all files or to save them all. At the bottom of the **File** menu appears a list of recently opened files. This makes it easier to open a file you are working on. (You can remove this list in the **Preferences**.)

Default, RCX Command Center saves backups of files you overwrite with an extension .BAK. This can be switched off in the **Preferences**.

Editing a File

The editor windows support all the usual edit functions. You can type in text, select it, delete it, etc. The editor color-codes the text such that it is easy to distinguish different aspects of the NQC program. Color-coding can be switched off in the **Preferences** item in the **Edit** menu.

In the **Edit** menu you also find the usual cut and paste commands that are common in all editors, with the usual shortcuts. Last changes can be undone.

Searching and Replacing

In the **Edit** menu you find commands for search for text. Fill in the text you want to search and select the appropriate options. For searching there is a corresponding button on the toolbar.

You can also replace text. Fill in the search and replace text and choose the options. You can either search and replace the occurrences one by one or all at once. For replacing there is a corresponding button on the toolbar.

There is also a command to go to a particular line in the text. Note that the current line and the position is always indicated in the status bar at the bottom of the window.

Templates

Templates are meant to avoid the work of typing pieces of code over and over again. At the right of the main window you find a small window with a number of items. Each such item is a template. (If you don't see the window, press once of twice on F9 to make it appear.) For example, there is one item that reads

```
task ..() { .. }
```

This is the task template. If you click on it, in your edit window the following appears:

```
task "name"()
{
  "statements"
}
```

The two words between double quotes, called fields, appear in red, and the first one is selected. These have to be replaced by your own code. Type the name of the task, e.g. t1. Next press F10 to move to the next field. Here you can type your statements. For example, you might want to switch a motor on. Choose the template

```
OnFwd(..);
```

In the edit window we now have the following text:

```
task t1()
{
  OnFwd("outs");

}
```

The first field is again selected. Type in the motor(s), and go to the next line to type the next statement. In this way you continue.

There are templates for language constructions and for the most important RCX commands. Using templates is fast and also avoids making errors. You can change the templates or add your own in the **Preferences**.

In the **Preferences** you can also indicate that you want the list of templates to appear as a popup menu when you press the right mouse button.

Macro's

Besides templates, RcxCC also offers macro's to quickly add pieces of text to your code. A macro consist of holding down the <Ctrl> and the <Alt> key (and sometimes the <Shift> key) and then pressing another key. For example, if you press <Ctrl><Alt>A the text OUT_A is added to your code. Macro's behave similar to templates and many templates are also available as macro's. Default the following macro's are defined:

<Ctrl><Alt>A	OUT_A
<Ctrl><Alt>B	OUT_B
<Ctrl><Alt>C	OUT_C
<Ctrl><Alt>1	SENSOR_1
<Ctrl><Alt>2	SENSOR_2
<Ctrl><Alt>3	SENSOR_3
<Ctrl><Alt>I	**if** construction
<Ctrl><Alt>H	**while** construction
<Ctrl><Alt>E	**else** construction
<Ctrl><Alt>F	OnFwd();
<Ctrl><Alt>R	OnRev();
<Ctrl><Alt>O	Off();
<Ctrl><Alt>L	Float();
<Ctrl><Alt>W	Wait();
<Ctrl><Alt><Shift>T	**task** definition
<Ctrl><Alt>><Shift>S	**sub** definition
<Ctrl><Alt>><Shift>V	**void** (= inline) definition
<Ctrl><Alt>><Shift>I	**int** definition

You can see a complete list of macro’s defined in the **Macro’s** tab in the **Preferences** window. Here you can also change the macro’s and add new macro’s.

Compiling, Downloading and Running

To compile the NQC program choose **Compile** from the **Compile** menu. This calls the NQC compiler, which is a separate process. When there are errors in your program you are notified and the error messages are shown below the edit window. Only a short description of each error is given. Clicking with your mouse on an error message will bring you to the corresponding line in the editor. You can remove the error panel by choosing **Hide Errors** in the **Compile** menu.

Note that the compiler provided with this version requires the program to be in NQC 2.0 format. This version of the language is not compatible with previous versions. If you have an old NQC program there are two things you can do. You can set the compiler to NQC 1.x compatibility mode in the Preference form. (This will not correct the color coding nor the new templates and macro’s.) Better, translate the program to NQC 2.0 by choosing **Translate 1.x to 2.0** in the **Edit** menu. This is not completely fool proof but works almost always correctly. Most translations are obvious. Only the commands Fwd and Rev are translated using two macro’s that are defined at the top of the program. Also a pragma is added that avoids initializing motors. (NQC 2.0 does that, NQC 1.x did not.)

If you want to see longer error messages, choose the **Show Code/Error Listing** option in the **Compile** menu. A form is shown with full descriptions of the errors. If there are no errors, this form will show a bytecode listing of the program. This is in general only useful for low-level debugging.

You can also immediately download the compiled program to the RCX. To this end, first select the program number in the toolbar or in the **Program Number** menu item. Next choose the **Download** command (or press the corresponding button on the toolbar). This runs the NQC compiler and (if there are no errors) downloads the code to the RCX. Because download also calls the compiler, there is no need to compile first.

If you also want to start the program immediately after downloading, choose **Download and Run**.

To run a program at a later stage, select the program number in the toolbar and choose **Run** (or the corresponding button on the toolbar). Stop the program by choosing **Stop**.

Direct Control

To directly control the RCX, choose the command **Direct Control** from the **Tools** menu. This opens a window with the following possibilities:

Sensors
Here you can set the mode and type of the three sensors. (For CyberMaster, only the mode can be set. The type shows which sensor you use.)

Motors
Here you can directly control the three motors of the robot. For each motor you have four options: Moving forwards, moving backwards, off (braking) and float (not braking). You can also set the speed of each motor.

Variables
Here you can set variables in the RCX. At the left choose the variable you like to change; in the middle, choose the operation (set, add, subtract, multiply, divide, logical and, logical or); and at the right choose the value with the up and down arrows, or type it in. Now press **Set** to make the change in the RCX.

Tasks
Here you can start and stop the 10 tasks (or 4 for the CyberMaster) that are in the current program in the RCX. If you start a task that is already running, it is restarted. Starting and stopping tasks is useful for testing your code and for making remotely controlled robots. For the second application you can also (and probably better) use the Joystick

Diagnostics

Clicking the **Diagnostics** menu item in the **Tools** menu opens a window displaying information about the RCX. Information you can find here includes whether the RCX is alive, the version of the firmware, the battery status, etc. (If the RCX is not alive, turn it on and press the **Refresh** button to recheck the information.)

In this window you can also set the level of the IR connection of the RCX, the watch, the power-down time, and what must be shown in the RCX display. (For the CyberMaster, most of these settings are not available.)

Watching the RCX

Clicking the **Watching the RCX** menu item in the **Tools** menu opens a window in which you can ask for the status of the sensors, motors, variables, etc, of the RCX. Indicate the items you want to poll by clicking on the checkmarks in front of them. (You can select or deselect all items by clicking on the appropriate buttons at the bottom.) Once you selected the items, press the **Poll Now** button to get the current status.

You can also continuously poll the information. To this end, select the time interval with the listbox at the right, and press the **Poll Regular** button. Please realize that polling the RCX takes quite some time. If you want to poll a lot of information regularly this will slow down both your computer and the robot.

RCX Piano

Choosing the **RCX Piano** menu item in the **Tools** menu, you can make the RCX play music. If you play with your mouse on the piano, the RCX makes the corresponding sounds. At the bottom left you can select the length of the notes. Add rests to your music using the **rest** button. The sequence of notes you play is stored. Press the **Play** button to play the whole sequence. Press **Clear** to clear the stored notes.

Pressing **Save** gives you the opportunity to save the collection of notes as an NQC program. You can load this program in the editor and download it to the robot. Finally clicking **Copy** copies the collection of notes as NQC code to the clipboard. In the editor you can paste it into you program. Note that RCX Piano defines two constant in you NQC code: __NOTETIME and __WAITTIME. You can change these to change the speed of the tune.

RCX Joystick

If you want to steer your robot with the computer as remote control, choose **RCX Joystick** in the **Tools** menu. RCX Joystick can work in two modes. The first mode assumes that two motors drive the left and right wheel of the robot. The second mode assumes one motor is used for driving and the other for steering. Indicate the correct. (The mode will be saved.) Below this you can indicate which motor is the left (or driving) motor, and which one it the right (or steering) motor. You can also indicate whether they should be reversed. Finally you can indicate the speed.

Now you are ready to move your robot. This can be done with joystick (if you have one), mouse, or keyboard. Moving with the joystick works as expected. . (If the robot moves in the wrong direction, check the settings.) The buttons on the joystick (1-2 or 1-4 depending on your joystick) start the corresponding tasks in the RCX. To use this, write a little NQC program that defines the first two (or four) tasks. (Let main just be a dummy task.) Download it but don’t start it. Now use the joystick to start the tasks.

To move the robot with the mouse, click on the buttons with the arrows. If you use the left mouse button, the robot moves in the direction as long as you keep the mouse button pressed. If you click with the right mouse button, the robot keeps on moving until you click another button. Clicking the buttons **T1** or **T2** starts task one or two in the robot. To move the robot with the keyboard, use the numeric keypad (make sure the NumLock key is pressed). As long as you keep the keys pressed, the robot move in the direction.
form, even if you don’t have a joystick.

Sending Messages

The RCX can react on messages. One RCX can send messages to another. But also the computer can send messages to the RCX. To use this feature, write an NQC program that looks at messages and makes to robot take action accordingly. For example, use the following program:

```
task main()
{
  while (true)
  {
    ClearMessage();
    until (Message() != 0);
    if (Message() == 1) {OnFwd(OUT_A+OUT_C);}
    if (Message() == 2) {OnRev(OUT_A+OUT_C);}
    if (Message() == 3) {Off(OUT_A+OUT_C);}
  }
}
```

It waits for a messages not equal to 0 and then takes action according to the number of the message. Now choose **Send Messages** from the **Tools** menu. Click on the right number button to send the corresponding message. If you want to send a message larger than 9, choose it with the up and down arrows and press the button labeled **Send**.

Please note that sending messages is relatively slow. It takes about half a second before the robot reacts.

Datalog

Not available in the CyberMaster.

The RCX supports a Datalog feature. The RCX can write data into an internal datalog which can then be uploaded to the computer. Creating the datalog and writing data to it is done by the program running in the RCX. See the NQC documentation on how to do this. Uploading the datalog needs to be done from the computer.

The **Datalog** item in the **Tools** menu open a window in which you can upload this datalog. Simply press the button labeled **Upload Datalog** and the contents of the datalog is shown.

You can also change the size of the datalog by choosing or typing in a size. Please realize that each item requires 3 bytes in the RCX. Because the memory of the RCX is rather small preferably make a small datalog. Changing the datalog size will erase its contents. The button **Clear Datalog** sets the size to 0.

You can save the contents of the datalog window to a file by pressing the button labeled **Save**.

Memory Map

Choosing the **Memory Map** item in the **Tools** menu, you can get information about the memory of the RCX. This is only useful for debugging when you experience weird problems in the RCX that might have to do with memory problems. In particular, the tool is useful if you suspect that you might not have enough memory for your programs.

For the MindStorms robots the following data is given:

- the starting addresses of the 8 subroutines in each program
- the starting addresses of the 10 tasks in each program
- the starting address of the datalog
- the address of the last datalog item
- the total amount of memory used
- the top of the user memory
- the amount of free memory left

For the CyberMaster you get:

- the starting addresses of the 4 subroutines
- the starting addresses of the 4 tasks
- the total amount of memory used
- the top of the user memory
- the amount of free memory left

Press the **Refresh** button to update the numbers (e.g. after you downloaded a program).

Clear Memory

Press the **Clear Memory** menu item in the **Tools** menu to clear the RCX memory. This command removes all tasks and subroutines in all programs, and removes the datalog (if any). Clearing memory is important when loading large programs because the RCX keeps all programs and tasks; also if you switch it off. After clearing memory you have about 6K for your program in the MindStorms robot (in the current version of the firmware). For the CyberMaster this is about 512 bytes.

Retransmissions Statistics

When the computer sends commands to the RCX it checks whether they arrive correctly. If they don't the command is retransmitted, that is, it is sent again. The CyberMaster also tries to do error correction. Errors occur when the robot is far away, or when there is lot of light (for the MindStorms) or noise (for the CyberMaster). To find out how well the transmissions work, choose **Retransmissions** from the **Tools** menu. A window is shown with the following information.

- The total number of commands transmitted
- The number of command that arrived correct (OK)

- The number of commands that arrived correct after 1 retransmission (1 RT)
- The number of commands that arrived correct after 2 retransmissions (2 RT)
- etcetera

On each line you see for numbers. At the left you find the numbers in the last second and at the right the numbers since the program started. In each case there are two numbers: the number of commands that arrived correct by themselves, and the number of commands that were correct after error correction (the second number is always 0 for the MindStorms).

Normally, all numbers should be empty except the ones on the top two lines. If this is not the case, better check for the MindStorms whether the IR tower is in far mode. If it is, either move the robot closer, place the IR Tower in a place where it is better seen, make sure that the IR connector of the robot is not covered, and avoid bright light in the room.

In this form you can also set the number of retries that are used when sending immediate commands (default 5) and when downloading (10). Setting them to 0 means that the computer does not wait for an answer from the robot. (Note that the download setting does not have effect on downloading NQC programs because this is done in a separate process, but is does have effect on downloading firmware.)

Find the RCX

If the program did not find the RCX at the startup, you can press the **Find RCX** menu item in the **Tools** menu to try again, e.g. after switching on the RCX. If you don't want to use the RCX anymore, you can choose **Turn RCX Off** from the **Tools** menu. This will turn the robot off and close the communication. If you want to use the RCX again later, turn it on and choose **Find RCX**.

You can also temporarily close the communication with the RCX, without turning the robot off, by choosing the **Close Communication** item in the **Tools** menu. This is useful if you want to run a different program that uses the RCX, but don't want to stop RcxCC. You can reopen the communication by clicking on **Reopen Communication**.

Download Firmware

Not available for the CyberMaster.

If the RCX looses its firmware (e.g. by changing the batteries) or if you want to download different firmware, use the **Download Firmware** command in the **Tools** menu. It will prompt you for the file containing the firmware. Next you need to have a little patience. A progress bar shows you how far the downloading is. (This is only an approximation because there is no way to actually check how far it is .). All programs are erased when downloading firmware. Make sure that the robot is close to the IR tower, because downloading firmware only works at short range.

Preferences

There are a number of preferences you can change. These will be saved between sessions. To this end, click the **Preferences** item in the **Edit** menu. A form with tabbed pages pops up. The following pages are available:

General
Here you can set some general properties. First of all, you can indicate whether you want that window positions are saved between sessions. You can also indicate whether or not backup copies are made of files that you are overwriting. (Backup copies have an extension .BAK.) Thirdly, you can indicate whether you want a list of recently opened files to be displayed in the **File** menu. Also you can indicate whether the toolbar and status bar should be visible. Finally, you can indicate whether the program should work in NQC 1.x compatibility mode. In this mode the compiler accepts NQC programs in older versions. (Note that color coding does not work correctly in this mode!) Press the **Default** button to return to the default settings. Press **OK** to save the changes. Press **Cancel** if you don't want to change them after all.

Startup

Here you can change the startup behavior. You can choose between showing the startup form (default), not trying to make a connection with the robot, or immediately making the connection. In the latter case you should specify the type of robot (MindStorms or CyberMaster) and the COM port. Press the **Default** button to return to the default settings. Press **OK** to save the changes. Press **Cancel** if you don't want to change them after all.

Editor
Here you can set some preferences concerning the editor. First of all you can indicate whether you want color-coding or not. Secondly, you can indicate whether the templates should also be put in a popup menu that appears when you click the right mouse button in the edit window. Also you can indicate whether or not you like the program to auto-indent lines of code. And you can switch off macro's such that <Ctrl><Alt> combinations are available for special characters. Finally, you can choose the font used in the edit window by clicking on the button **Font**. (Note that only font name and size have effect. The style is dependent on the syntax; e.g. italic for comments.) Press the **Default** button to return to the default settings. Press **OK** to save the changes. Press **Cancel** if you don't want to change them after all.

Templates
Here you can change the templates. You see a list of all templates currently defined. Click **Insert** to insert a new template above the selected one. Click **Change** to change the selected template and click **Delete** to delete it. You can also double click on a template to change it. Use the arrows to move the current selected template up or down in the list. Each template is defined on one line. A line that consists of a single – symbol means a horizontal separator. A | symbol represents a vertical separator between columns in the menu. Other lines are the actual templates. Write fields between double quotes. The \ symbol is given by \\. Go to the next line with \=; to the next line with an indentation with \>; and the next line going out of an indentation with \<. So for example:

```
if ("condition")\={\>"statements"\<}\=else\={\>"statements"\<}\=
```

results in

```
if ("condition")
{
  "statements"
}
else
{
  "statements"
}
```

Once you are done, press **OK** to save the changes. Press **Cancel** if you don't want to change them after all. Press the **Default** button to return to the default templates.

Macro's
Here you can change the macro's and define your own. There are two lists of macro's. One without the <Shift> key pressed, and the other with the <Shift> key press. To switch between the two lists, click on the **Shift** checkmark. You can define macro's for each of the 26 letters and for the number 0 to 9. So in total you can define 72 macro's. To change or define a macro, double click on it, or select it and press the **Change** button. To remove a macro, select it and press the **Delete** button. Macro's are defined in exactly the same way as templates (see above). An undefined macro will give the character the combination stands for. This is important for people having keyboards on which certain characters (like brackets) are not available. You might want to delete certain macros in this case. Once you are done, press **OK** to save the changes. Press **Cancel** if you don't want to change the macro's after all. Press the **Default** button to return to the default macros.

NQC Statements

Here is a quick overview of NQC statements. For more information see the NQC guide or the tutorial.

Statement	**Description**
while (*cond*) *body*	Execute body zero or more times while condition is true
do *body* while (*cond*)	Execute body one or more times while condition is true
until (*cond*) *body*	Execute body zero or more times until condition is true
switch (*expression*) *body*	Execute alternatives within *body* based on value of *expression*
break	Break out from while/do/until body
continue	Skip to next iteration of while/do/until body
repeat (*expression*) *body*	Repeat body a specified number of times

if (*cond*) *stmt1*
if (*cond*) *stmt1* **else** *stmt2*
	Execute stmt1 if condition is true. Execute stmt2 (if present) if condition is false.

start *task_name*	Start the specified task
stop *task_name*	Stop the specified task
function(*args*)	Call a function using the supplied arguments
var = *expression*	Evaluate expression and assign to variable
var += *expression*	Evaluate expression and add to variable
var -= *expression*	Evaluate expression and subtract from variable
var *= *expression*	Evaluate expression and multiply into variable
var /= *expression*	Evaluate expression and divide into variable
var |= *expression*	Evaluate expression and perform bitwise OR into variable
var &= *expression*	Evaluate expression and perform bitwise AND into variable
return	Return from function to the caller
expression	Evaluate expression

NQC Conditions

Here is a quick overview of NQC conditions. For more information see the NQC guide or the tutorial. Conditions are used within control statements to make decisions. In most cases, the condition will involve a comparison between expressions.

Condition	**Meaning**
true	always true
false	always false
`expr1 == expr2`	test if expressions are equal
`expr1 != expr2`	test if expressions are not equal
`expr1 < expr2`	test if one expression is less than another
`expr1 <= expr2`	test if one expression is less than or equal to another
`expr1 > expr2`	test if one expression is greater than another
`expr1 >= expr2`	test if one expression is greater than or equal to another
`! condition`	logical negation of a condition
`cond1 && cond2`	logical AND of two conditions (true if and only if both conditions are true)
`cond1 || cond2`	logical OR of two conditions (true if and only if at least one of the conditions are true)

NQC Expressions

Here is a quick overview of NQC expresions. For more information see the NQC guide or the tutorial. There are a number of different values that can be used within expressions including constants, variables, and sensor values. Note that SENSOR_1, SENSOR_2, and SENSOR_3 are macros that expand to SensorValue(0), SensorValue(1), and SensorValue(2) respectively.

Value	Description
`number`	A constant value (e.g. "123")
`variable`	A named variable (e.g "x")
`Timer(n)`	Value of timer n, where n is between 0 and 3
`Random(n)`	Random number between 0 and n
`SensorValue(n)`	Current value of sensor n, where n is between 0 and 2
`Watch()`	Value of system watch
`Message()`	Value of last received IR message

Values may be combined using operators. Several of the operators may only be used in evaluating constant expressions, which means that their operands must either be constants, or expressions involving nothing but constants. The operators are listed here in order of precedence (highest to lowest).

Operator	Description	Associativity	Restriction	Example

abs()
sign()
	Absolute value
	Sign of operand
		n/a
		n/a
				`abs(x)`
				`sign(x)`

`++`
`--`
	Increment
	Decrement
		left
		left
			variables only
			variables only
				`x++ or ++x`
				`x-- or --x`

`-` `~`	Unary minus Bitwise negation (unary)	right right	constant only	`-x` `~123`

`*`
`/`
`%`
	Multiplication
	Division
	Modulo
		left
		left
		left
			constant only	`x * y`
				`x / y`
				`123 % 4`

`+`
`-`
	Addition
	Subtraction
		left
		left
				`x + y`
				`x - y`

`<<`
`>>`
	Left shift
	Right shift
		left
		left
			constant only
			constant only
				`123 << 4`
				`123 >> 4`

`&`	Bitwise AND	left		`x & y`
`^`	Bitwise XOR	left	constant only	`123 ^ 4`
`|`	Bitwise OR	left		`x | y`
`&&`	Logical AND	left	constant only	`123 && 4`
`||`	Logical OR	left	constant only	`123 || 4`

NQC Functions

Here is a quick overview of NQC functions. For more information see the NQC guide or the tutorial. Most of the functions require all arguments to be constant expressions (number or operations involving other constant expressions). The exceptions are functions that use a sensor as an argument, and those that can use any expression. In the case of sensors, the argument should be a sensor name: SENSOR_1, SENSOR_2, or SENSOR_3. In some cases there are predefined names (e.g. SENSOR_TOUCH) for appropriate constants.

Function	Description	Example
`SetSensor(sensor, config)`	Configure a sensor.	`SetSensor(SENSOR_1, SENSOR_TOUCH)`
`SetSensorMode(sensor, mode)`	Set sensor's mode	`SetSensor(SENSOR_2, SENSOR_MODE_PERCENT)`

`SetSensorType(sensor, type)`	Set sensor's type	`SetSensor(SENSOR_2, SENSOR_TYPE_LIGHT)`
`ClearSensor(sensor)`	Clear a sensor's value	`ClearSensor(SENSOR_3)`
`On(outputs)`	Turn on one or more outputs	`On(OUT_A + OUT_B)`
`Off(outputs)`	Turn off one or more outputs	`Off(OUT_C)`
`Float(outputs)`	Let the outputs float	`Float(OUT_B)`
`Fwd(outputs)`	Set outputs to forward direction	`Fwd(OUT_A)`
`Rev(outputs)`	Set outputs to backwards direction	`Rev(OUT_B)`
`Toggle(outputs)`	Flip the direction of outputs	`Toggle(OUT_C)`
`OnFwd(outputs)`	Turn on in forward direction	`OnFwd(OUT_A)`
`OnRev(outputs)`	Turn on in reverse direction	`OnRev(OUT_B)`
`OnFor(outputs, time)`	Turn on for specified number of 100ths of a second. Time may be an expression.	`OnFor(OUT_A, 200)`
`SetOutput(outputs, mode)`	Set output mode	`SetOutput(OUT_A, OUT_ON)`
`SetDirection(outputs, dir)`	Set output direction	`SetDirection(OUT_A, OUT_FWD)`
`SetPower(outputs, power)`	Set output power level (0-7). Power may be an expression.	`SetPower(OUT_A, 6)`
`Wait(time)`	Wait for the specified amount of time in 100ths of a second. Time may be an expression.	`Wait(x)`
`PlaySound(sound)`	Play the specified sound (0-5).	`PlaySound(SOUND_CLICK)`
`PlayTone(freq, duration)`	Play a tone of the specified frequency for the specified amount of time (in 10ths of a second)	`PlayTone(440, 5)`
`ClearTimer(timer)`	Reset timer (0-3) to value 0	`ClearTimer(0)`
`StopAllTasks()`	Stop all currently running tasks	`StopAllTasks()`
`SelectDisplay(mode)`	Select one of 7 display modes: 0: system watch, 1-3: sensor value, 4-6: output setting. Mode may be an expression.	`SelectDisplay(1)`
`SendMessage(message)`	Send an IR message (1-255). Message may be an expression.	`SendMessage(x)`
`ClearMessage()`	Clear the IR message buffer	`ClearMessage()`
`CreateDatalog(size)`	Create a new datalog of the given size	`CreateDatalog(100)`
`AddToDatalog(value)`	Add a value to the	`AddToDatalog(Timer(0))`

	datalog. The value may be an expression.	
`SetWatch(hours, minutes)`	Set the system watch value	`SetWatch(1,30)`
`SetTxPower(hi_lo)`	Set the infrared transmitter power level to low or high power	`SetTxPower(TX_POWER_LO)`

NQC Constants

Here is a quick overview of NQC constants. For more information see the NQC guide or the tutorial. Many of the values for RCX functions have named constants that can help make code more readable. Where possible, use a named constant rather than a raw value.

Sensor configurations for SetSensor()	`SENSOR_TOUCH, SENSOR_LIGHT, SENSOR_ROTATION, SENSOR_CELSIUS, SENSOR_FAHRENHEIT, SENSOR_PULSE, SENSOR_EDGE`
Modes for SetSensorMode()	`SENSOR_MODE_RAW, SENSOR_MODE_BOOL, SENSOR_MODE_EDGE, SENSOR_MODE_PULSE, SENSOR_MODE_PERCENT, SENSOR_MODE_CELSIUS, SENSOR_MODE_FAHRENHEIT, SENSOR_MODE_ROTATION`
Types for SetSensorType()	`SENSOR_TYPE_TOUCH, SENSOR_TYPE_TEMPERATURE, SENSOR_TYPE_LIGHT, SENSOR_TYPE_ROTATION`
Outputs for On(), Off(), etc.	`OUT_A, OUT_B, OUT_C`
Modes for SetOutput()	`OUT_ON, OUT_OFF, OUT_FLOAT`
Directions for SetDirection()	`OUT_FWD, OUT_REV, OUT_TOGGLE`
Output power for SetPower()	`OUT_LOW, OUT_HALF, OUT_FULL`
Sounds for PlaySound()	`SOUND_CLICK, SOUND_DOUBLE_BEEP, SOUND_DOWN, SOUND_UP, SOUND_LOW_BEEP, SOUND_FAST_UP`
Modes for SelectDisplay()	`DISPLAY_WATCH, DISPLAY_SENSOR_1, DISPLAY_SENSOR_2, DISPLAY_SENSOR_3, DISPLAY_OUT_A, DISPLAY_OUT_B, DISPLAY_OUT_C`
Tx power level for SetTxPower()	`TX_POWER_LO, TX_POWER_HI`