UCAS C Programming Guide

The Basics

task main() // initializes a task, basically just put this at the beginning (unless integers are assigned)  
{

} // write all code between these brackets.

; // at the end of all commands (highlighted in blue) place a ; (shown below)

OnFwd(OUT\_A, Pwr%); // code to move engine forward [A=motor, % is a number ex. 75]

OnRev(OUT\_AC, 60); // same but for reverse, notice this turns on 2 motors (A, and B)

Wait(4000); // Wait for this amount of time (in ms, thus 4000 = 4 seconds) before continuing

/note on wait, wait continues the previous lines running until the end of wait. If you don’t place a wait after a Fwd or Rev (or anything for that matter) it will continue with the next command (if no other the program ends, shutting off the move line).

Off(OUT\_BC); // turns off the motor (power off and brakes)

Float(OUT\_B); // turns motor off (shuts off power)

More complexity

#define Rotate 1000 // this creates a set # called Rotate with the value 1000, Placed at beginning of program before task main()

/note, #define sometimes doesn’t work (bugs) with certain versions of NXC

If I write Wait(Rotate); it would wait for 1 second, if #define is used properly.

int x=1000; // defines a variable, remember to place before “task main()” (all variables)

X += 200; // adds 200 to variable X

X -= 200; X \*= 2;, X /=;, // for subtract, multiply or divide.

Random(600); // generates random number from 0-600 (i.e. X = Random(300); or Wait(Random(600)).)

4 different switch and control statements

Repeat (4)  
{  
} // Repeat the part of the program in brackets 4 times (can also contain variables)

While(true)

{

} // infinite loop (in place of “true” can use variable (i.e. While(X=13))

**do**

{

}

**while** (condition); // condition is “true” (always) or a variable.

until(); // continue previous command until inserted is true. Like Sensor\_1==1, or ttt != 35

Sensors

== equal to

< smaller than

<= smaller than or equal to

> larger than

>= larger than or equal to

!= not equal to

**true** always true

**false** never true

ttt != 3 true when ttt is not equal to 3

(ttt >= 5) && (ttt <= 10) true when ttt lies between 5 and 10

(aaa == 10) || (bbb == 10)true if either aaa or bbb (or both) are equal to 10

SetSensorSound(IN\_1); // Sets Sensor 1 to Sound Sensor

SetSensor(IN\_3, SENSOR\_TOUCH); // alternate way to set sensor

SetSensorMode(IN\_3, SENSOR\_MODE\_BOOL); // affects how sensor works

-BOOL=0-1  
-PERCENT= 0-100%  
-EDGE = counts the amount of positive(numerical) changes  
-Pulse= counts amount of changes

ClearSensor(); // clears counters of edge or pulse on a sensor

The more complex sensors

SetSensorType(IN\_3,IN\_TYPE\_LIGHT\_INACTIVE); // light sensor (type is optional)

// if you write ACTIVE instead of INACTIVE, it makes it a lamp rather than a sensor

SetSensorLowspeed(IN\_4); //(may also have “\_9V” after lowspeed) ultrasonic sensor

//To read Lowspeed sensor, you must write “SensorUS(IN\_#)”

Music

PlayToneEx(245,1700,70,FALSE); // (frequency, duration, volume, loop?)

Drawing

ClearScreen() clears the screen;

NumOut(x, y, number) lets you specify coordinates, and number;

TextOut(x, y, string) works as above, but outputs a text string

GraphicOut(x, y, filename) shows a bitmap **.ric** file

CircleOut(x, y, radius) outputs a circle specified by the coordinates of the center and radius;

LineOut(x1, y1, x2, y2) draws a line that goes from point (x1,x2) to (x2,y2)

PointOut(x, y) puts a dot on the screen

RectOut(x, y, width, height) draws a rectangle with the bottom left vertex in (x,y) and with the

dimensions specified;

ResetScreen() resets the screen.

GraphicOut(x,y, “filename”, loop?) // display a graphic from within the brick