Reserved Words

Motors

Motor control and some fine-tuning commands.

`motor[output] = power;`
This turns the referenced NXT motor output either on or off and simultaneously sets it's power level. The NXT has 3 motor outputs: `motorA, motorB, and motorC`. The NXT supports power levels from -100 (full reverse) to 100 (full forward). A power level of 0 will cause the motors to stop.

```
motor[motorC]= 100;    //motorC - Full speed forward
motor[motorB]= -100;   //motorB - Full speed reverse
```

`bMotorFlippedMode[output] = 1;` (or `0;`)
When set equal to one, this code reverses the rotation of the referenced motor. Once set, the referenced motor will be reversed for the entire program (or until bMotorFlippedMode[] is set equal to zero).

This is useful when working with motors that are mounted in opposite directions, allowing the programmer to use the same power level for each motor.

There are two settings: `0` is normal, and `1` is reverse.

Before:

```
motor[motorC]= 100;     //motorC - Full speed forward
motor[motorB]= 100;     //motorB - Full speed forward
```

After:

```
bMotorFlippedMode[motorC]= 1; //Flip motor C's direction
motor[motorC]= 100;           //motorC - Full speed reverse
motor[motorB]= 100;           //motorA - Full speed forward
```

`bFloatDuringInactiveMotorPWM = true;` (or `false;`)
This is used to set whether the motors on the NXT will float or brake when there is no power applied.

There are two settings:

`false` - motors will brake when inactive

`true` - motors will float when inactive

```
bFloatDuringInactiveMotorPWM = false;
          //motors will break when power is set to 0
```

Reserved Words

`nMotorPIDSpeedCtrl[output] = mtrSpeedReg;`
This line of code enables the PID control for the specified motor output. The PID control adjusts the actual amount of power sent to a motor to match the desired value, specified in the program. Each motor that needs to be regulated must be referenced using an instance of this code.

The NXT has 3 motor outputs: `motorA`, `motorB`, and `motorC`.

```
nMotorPIDSpeedCtrl[motorC] = mtrSpeedReg;
    //PID control on motorC enabled
nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;
    //PID control on motorB enabled
motor[motorC] = 50; //motorC adjusts to spin at 50% speed...
                    //...even when friction varies
motor[motorB] = 50; //motorB adjusts to spin at 50% speed...
                    //...even when friction varies
wait1Msec(4000);    //wait for 4 seconds
```

`nSyncedMotors = synch_type;`
This line of code synchronizes the power level of one motor to another, specified in the synch_type.

The common configurations are: `synchAB`, `synchAC`, `synchBA`, `synchBC`, `synchCA`, and `synchCB`.

The first capital letter of the synch_type refers to the "Master" motor and the second capital letter refers to the "Slave" motor. The "Slave" motor basis it's behavior after the "Master" motor.

```
nSyncedMotors = synchBC; //sets motorC to imitate motorB
```

`nSyncedTurnRatio = percentage;`
This line of code establishes the relationship between the motors referened in the nSyncedMotors command. Percentage values range from -100 to 100, with the "Slave" motor doing the exact opposite of the "Master" motor (-100), to the "Slave" perfectly imitating the "Master" (100).

```
nSyncedMotors = synchBC; //sets motorC to imitate motorB
nSyncedTurnRatio = 100;
motor[motorB] = 50;      //both motorB and motorC move...
                         //...forward at half power
wait1Msec(4000);         //wait for 4 seconds
```

Reserved Words

Timing

The NXT allows you to use Wait commands to insert delays into your program. It also supports Timers, which work like stopwatches; they count time, and can be reset when you want to start or restart tracking time elapsed.

`wait1Msec(wait_time);`
This code will cause the robot to wait a specified number of milliseconds before executing the next instruction in a program. "wait_time" is an integer vlaue (where 1 = 1/1000th of a second). Maximum wait_time is 32768, or 32.768 seconds.

```
motor[motorA]= 100;      //motorA - full speed forward
wait1Msec(2000);         //Wait 2 seconds
motor[motorA]= 0;        //motorA - off
```

`wait10Msec(wait_time);`
This code will cause the robot to wait a specified number of hundredths of seconds before executing the next instruction in a program. "wait_time" is an integer vlaue (where 1 = 1/100th of a second). Maximum wait_time is 32768, or 327.68 seconds.

```
motor[motorA]= 100;      //motorA - full speed forward
wait10Msec(200);         //Wait 2 seconds
motor[motorA]= 0;        //motorA - off
```

`time1[timer]`
This code returns the current value of the referenced timer as an integer. The resolution for "time1" is in milliseconds (1 = 1/1000th of a second).

The maximum amount of time that can be referenced is 32.768 seconds (~1/2 minute)

The NXT has 4 internal timers: `T1`, `T2`, `T3`, and `T4`

```
int x;           //Integer variable x
x=time1[T1];     //Assigns x=value of Timer 1 (1/1000 sec.)
```

`time10[timer]`
This code returns the current value of the referenced timer as an integer. The resolution for "time10" is in hundredths of a second (1 = 1/100th of a second).

The maximum amount of time that can be referenced is 327.68 seconds (~5.5 minutes)

The NXT has 4 internal timers: `T1`, `T2`, `T3`, and `T4`

```
int x;           //Integer variable x
x=time10[T1];    //Assigns x=value of Timer 1 (1/100 sec.)
```

Reserved Words

`time100[timer]`
This code returns the current value of the referenced timer as an integer. The resolution for "time100" is in tenths of a second (1 = 1/10th of a second).

The maximum amount of time that can be referenced is 3276.8 seconds (~54 minutes)

The NXT has 4 internal timers: `T1`, `T2`, `T3`, and `T4`

```
int x;           //Integer variable x
x=time100[T1];   //assigns x=value of Timer 1 (1/10 sec.)
```

`ClearTimer(timer);`
This resets the referenced timer back to zero seconds.

The NXT has 4 internal timers: `T1`, `T2`, `T3`, and `T4`

```
ClearTimer(T1); //Clear Timer #1
```

Sensors

Sensor commands for configuration and usage are listed below. Most sensor setup should be done through the Robot > Motors and Sensors Setup menu for best results.

`SetSensorType(sensor_input,sensor_type);`
This function is used to manually set the mode of a specific input port to a specific type of sensor. We recommend, however, that you use the "Motor and Sensors Setup" wizard in ROBOTC.

The NXT has 4 sensor inputs: `S1`, `S2`, `S3`, and `S4` and supports 8 different types of sensors:

Sensor Type	Type	Description	Range of Values
sensorTouch	RCX &NXT	Digital	0 to 1
sensorTemperature	RCX	Analog Temperature	0.0 t 100.0
sensorReflection	RCX	Analog Percentage	0 to 100
sensorRotation	RCX	Digital with Directional counter	-32768 to 32768
sensorLightActive	NXT	Analog, Percentage (Light Sensor with LED)	0 to 100
sensorLightInactive	NXT	Analog, Percentage (Light Sensor w/out LED)	0 to 100
sensorSoundDB	NXT	Analog, Percentage	0 to 100
sensorSONAR	NXT	Distance, CM	0 to 255

```
SetSensorType(S1,sensorTouch); //Input 1 set as a touch sensor
```

Reserved Words

`SensorValue(sensor_input)`
SensorValue is used to reference the integer value of the specified sensor port. Values will correspond to the type of sensor set for that port.

The NXT has 4 sensor inputs: `S1`, `S2`, `S3`, and `S4`

```
SetSensorType(S1, sensorTouch);//Input 1 set as touch sensor
if(SensorValue(S1) == 1)    //If the touch sensor is pressed
{
  motor[motorA] = 100;      //motorA - full speed forward
}
```

`nMotorEncoder[motor]`
This code is used to access the internal encoder from the NXT's motors. An integer value is returned with the number of degrees the motor has traveled (1 = 1 degree).

```
while(nMotorEncoder[motorC]<100)//While motorC encoder has
{                               //spun less than 100 degrees
  motor[motorC] = 100;      //motorC - full speed forward
  motor[motorB] = 100;      //motorB - full speed forward
}
```

You can also assign the value of nMotorEncoder to 0 to reset the encoder.

```
nMotorEncoder[motorA]=0;  //motorA encoder is set to
```

`ClearSensorValue(sensor_input);`
This function resets the value of the referenced sensor port back to zero. This is only neccessary with specific sensor types that retain their values (e.g. Encoder).

The NXT has 4 sensor inputs: `S1`, `S2`, `S3`, and `S4`

```
SetSensorType(S1,sensorRotation);//Input 1 set to
                                 //Rotation Sensor.
ClearSensorValue(S1);            //Reset Input #1 back to 0
```

Sounds

The NXT can generate tones or play stored waveform sound data.

`PlayTone(frequency, duration);`
This plays a sound from the NXT internal speaker at a specific frequency (1 = 1 hertz) for a specific length (1 = 1/100th of a second).

```
PlayTone(220, 599); //Plays a 220hz tone for 1/2 second
```

Reserved Words

`PlaySound(sound_name);`
Plays a sound effect from the NXT internal library. Requires a sound name to be passed to play the sound. The sound names are:

- `soundBlip`
- `soundBeepBeep`
- `soundDownwardTones`
- `soundUpwardTones`
- `soundLowBuzz`
- `soundFastUpwardTones`
- `soundShortBlip`
- `soundException`

```
PlaySoundTone(soundUpwardTones);//Plays "Upward Tones" sound
```

`PlaySoundFile(file_name);`
This function is used to play a sound file that is on the NXT. NXT sounds files have the .rso extension.

```
PlaySoundFile(Whoops.rso); //Plays the file "Whoops.rso"
```

LCD Display
Commands for the NXT's LCD Display.

`nxtDisplayStringAt(xPosition, yPosition, text, var1, var2, var3);`
Displays a text line on the NXT's LCD screen. Up to three variables are passable to the function.

`xPosition` - This integer value is the number of pixels away from the **left** of the display that you want your string to be printed at.
`yPosition` - This integer value is the number of pixels away from the **bottom** of the display that you want your string to be printed at. Leaving this value at 0 will cause any characters to be cut off.
`text` - The text parameter is what shows up on the screen. This will be a string enclosed in quotes up to 16 characters. You may also display up to 3 variables in this parameter by adding %d up to three times. Remember that you can only display 16 total characters, so the value of the variables will take up some of those 16 characters.
`var1, var2, var3` - These (optional) parameters define which variables will be displayed to the screen and each must correspond to a separate %d within the text parameter.

```
int x = 1;    //declares first variable
int y = 2;    //declares second variable
int z = 3;    //declares third variable
nxtDisplayStringAt(0,31,"Test %d %d %d" ,x,y,z);
              //Displays "Test 123"
```

`eraseDisplay();`
Clears the NXT's LCD screen of all text and GUI images.

```
eraseDisplay(); //Clears NXT screen of all images and text
```

Reserved Words

Miscellaneous

Miscellaneous useful commands that are not part of the standard C language.

`srand(seed);`
Defines the integer value of the "seed" used in the random() command to generate a random number. This command is optional when using the random() command, and will cause the same sequence of numbers to be generated each time that the program is run.

```
srand(16);   //Assign 16 as the value of the seed
```

`random(value);`
Generates random number between 0 and the number specified in its parenthesis.

```
random(100);   //Generates a number between 0 and 100
```

Control Structures

Program control structures in ROBOTC enable a program to control its flow outside of the typical left to right and top to bottom fashion.

`task main(){}`
Creates a task called "main" needed in every program. Task main is responsible for holding the code to be executed within a program.

`while(condition){}`
Used to repeat a {section of code} while a certain (condition) remains true. Infinite while loops can be created by ensuring that the condition is always true, e.g. "1==1" or "true".

```
while(time1[T1]<5000)//While the timer is less than 5 sec...
{
 motor[motorA]= 100;//...motorA runs at 100%
}
```

`if(condition){}/else{}`
With this command, the program will check the (condition) within the if statement's parentheses and then execute one of two sets of code. If the (condition) is true, the code inside the if statement's curly braces will be run. If the (condition) is false, the code inside the else statement's curly braces will be run instead.

```
if(sensorValue(touch) ==1)//the touch sensor is used as...
{                          //...the condition
 motor[motorA]= 0;   //if it's pressed motorA stops
}
else
{
  motor[motorA]= 100; //if it's not pressed motorA runs
}
```

Reserved Words

Data Types

Different types of information require different types of variables to hold them.

`int`
This data type is used to store integer values ranging from -32768 to 32768.

```
int x;   //Declares the integer variable x
x = 765; //Stores 765 inside of x
```

The code above can also be written:

```
int x = 765;  //Declares the integer variable x and...
              //...initializes it to a value of 765
```

`long`
This data type is used to store integer values ranging from -2147483648 to 2147483648.

```
long x;       //Declares the long integer variable x
x = 76543210; //Stores 76543210 inside of x
```

`float`
This data type is used to store decimal or floating point numbers.

```
float x;      //Declares the float variable x
x = 77.932;   //Stores 77.932 inside of x
```

`bool`
This data type is used to store boolean values of either 1 (also true) or 0 (also false).

```
bool x;       //Declares the bool variable x
x = 0;        //Sets x to 0
```

`char`
This data type is used to store a single character, specified between a set of single quotes.

```
char x;       //Declares the char variable x
x = 'J';      //Stores the character J inside of x
```

`string`
This data type is used to store a string of characters, such as a word or sentence, specified between a set of double quotes.

```
string x;             //Declares the long integer variable x
x = "ROBOTC rocks!"; //Stores ROBOTC rocks! inside of x
```