

# MaSH Environment rcx

Andrew Rock  
School of Information and Communication Technology  
Griffith University  
Nathan, Queensland, 4111, Australia  
a.rock@griffith.edu.au

June 9, 2011

## Contents

<b>1 Purpose</b>	<b>2</b>
<b>2 Rewrites</b>	<b>2</b>
<b>3 Setting up sensors</b>	<b>2</b>
3.1 Constants . . . . .	2
3.2 Methods . . . . .	3
<b>4 Using touch sensors</b>	<b>3</b>
4.1 Methods . . . . .	3
<b>5 Using light sensors</b>	<b>4</b>
5.1 Methods . . . . .	4
<b>6 Using rotation sensors</b>	<b>5</b>
6.1 Methods . . . . .	5
<b>7 Output port constants</b>	<b>5</b>
7.1 Constants . . . . .	5
<b>8 Using motors</b>	<b>6</b>
8.1 Methods . . . . .	6
<b>9 Using lamps</b>	<b>6</b>
9.1 Methods . . . . .	6
<b>10 Waiting for fixed times</b>	<b>7</b>
10.1 Methods . . . . .	7

<b>11 Making sounds</b>	<b>7</b>
11.1 Methods . . . . .	7
<b>12 Using the LCD</b>	<b>7</b>
12.1 Methods . . . . .	7
<b>13 Using infra-red communications</b>	<b>8</b>
13.1 Methods . . . . .	8
<b>14 Math</b>	<b>8</b>
14.1 Purpose . . . . .	8
14.2 Constants . . . . .	8
14.3 Methods . . . . .	9
<b>15 Strings</b>	<b>10</b>
15.1 Purpose . . . . .	10
15.2 Methods . . . . .	10

# 1 Purpose

This environment supports programming a Lego Mindstorms RCX robot, via the Lejos system.

# 2 Rewrites

*mandatory*

```
void main ()
```

*Purpose:* A program that is organised into methods must have a main method (a procedure with no arguments). This will be the first method to execute. `mashc` automatically rewrites this method to conform to standard Java.

# 3 Setting up sensors

## 3.1 Constants

```
final int TOUCH
```

*Purpose:* Constant to select sensor type touch.

```
final int LIGHT
```

*Purpose:* Constant to select sensor type light.

`final int ROTATION`

*Purpose:* Constant to select sensor type rotation.

## 3.2 Methods

`void setUpSensor (int port, int type)`

*Purpose:* Sets up the `port` to be sensor of the given `type`.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* `type` is TOUCH, LIGHT, or ROTATION.

# 4 Using touch sensors

## 4.1 Methods

`void waitForPush (int port)`

*Purpose:* Makes the program wait until the touch sensor on `port` is pushed.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a touch sensor.

`void waitForLetGo (int port)`

*Purpose:* Makes the program wait until the touch sensor on `port` is let go.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a touch sensor.

`boolean isPushed (int port)`

*Purpose:* Returns `true` if and only if the button on `port` is currently pushed.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a touch sensor.

## 5 Using light sensors

### 5.1 Methods

`void waitForLighter (int port, int dif)`

*Purpose:* Makes the program wait until the light sensor reading on `port` is increased by `dif`.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a light sensor.

*Precondition:* `dif` is between 0 and 100, inclusive. 0 is no wait at all. Real light levels never really change by anything like 100.

`void waitForLight (int port, int light)`

*Purpose:* Makes the program wait until the light sensor reading on `port` is at least the desired `light` level.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a light sensor.

*Precondition:* `light` is between 0 and 100, inclusive.

`void waitForDarker (int port, int dif)`

*Purpose:* Makes the program wait until the light sensor reading on `port` is decreased by `dif`.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a light sensor.

*Precondition:* `dif` is between 0 and 100, inclusive. 0 is no wait at all. Real light levels never really change by anything like 100.

`void waitForDark (int port, int light)`

*Purpose:* Makes the program wait until the light sensor reading on `port` is at most the desired `light` level.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a light sensor.

*Precondition:* `light` is between 0 and 100, inclusive.

`int getLight (int port)`

*Purpose:* Returns the current light sensor reading on `port`.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a light sensor.

## 6 Using rotation sensors

### 6.1 Methods

`void waitForRotation (int port, int rotation)`

*Purpose:* Makes the program wait until the counter in the rotation sensor on `port` has changed by at least the absolute value of `rotation`.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a rotation sensor.

`int getRotation (int port)`

*Purpose:* Returns the current rotation sensor reading on `port`.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a rotation sensor.

`void resetRotation (int port)`

*Purpose:* Sets the counter in the rotation sensor on `port` to zero.

*Precondition:* `port` is 1, 2, or 3.

*Precondition:* The port has been set up as a rotation sensor.

## 7 Output port constants

### 7.1 Constants

`final int A`

*Purpose:* Constant to select port A.

`final int B`

*Purpose:* Constant to select port B.

`final int C`

*Purpose:* Constant to select port C.

## 8 Using motors

### 8.1 Methods

`void motorForward (int port, int power)`

*Purpose:* Make the motor on `port` go forwards at the given `power`.

*Precondition:* `port` is A, B, or C.

*Precondition:* `power` is between 0 and 7, inclusive.

`void motorBackward (int port, int power)`

*Purpose:* Make the motor on `port` go backwards at the given `power`.

*Precondition:* `port` is A, B, or C.

*Precondition:* `power` is between 0 and 7, inclusive.

`void motorStop (int port)`

*Purpose:* Stop the motor on `port`.

*Precondition:* `port` is A, B, or C.

`void motorFloat (int port)`

*Purpose:* Float the motor on `port`.

*Precondition:* `port` is A, B, or C.

## 9 Using lamps

### 9.1 Methods

`void lampOn (int port, int power)`

*Purpose:* Make the lamp on `port` go on at the given `power`.

*Precondition:* `port` is A, B, or C.

*Precondition:* `power` is between 0 and 7, inclusive.

`void lampOff (int port)`

*Purpose:* Turns the lamp on `port` off.

*Precondition:* `port` is A, B, or C.

## 10 Waiting for fixed times

### 10.1 Methods

```
void sleep (int ms)
```

*Purpose:* Makes the program wait for a requested number of ms (milliseconds).

## 11 Making sounds

### 11.1 Methods

```
void systemSound (int i)
```

*Purpose:* Play system sound number *i*. The system sounds are as follows.

<i>i</i>	<i>description</i>
0	short beep
1	double beep
2	descending arpeggio
3	ascending arpeggio
4	long, low beep
5	quick ascending arpeggio

*Precondition:*  $0 \leq i \leq 5$ .

```
void playTone (int frequency, int duration)
```

*Purpose:* Plays a tone, given its frequency (Hertz) and duration (centiseconds).

*Precondition:*  $31 \leq \text{frequency} \leq 2100$ .

*Precondition:*  $0 \leq \text{duration} \leq 256$ .

## 12 Using the LCD

### 12.1 Methods

```
void showNumber (int a)
```

*Purpose:* Displays a number in the LCD. Does not require `refresh()`.

*Precondition:*  $0 \leq a \leq 9999$ .

```
void clear ()
```

*Purpose:* Clears the LCD, but the effect will not show until `refresh()` is called.

```
void refresh ()
```

*Purpose:* Refreshes the LCD, causing the changes to be displayed.

```
void putChar (char c, int i)
```

*Purpose:* Puts a character `c` into the LCD at position `i`.

*Precondition:*  $0 \leq i \leq 4$ .

## 13 Using infra-red communications

### 13.1 Methods

```
void sendByte (int i)
```

*Purpose:* Send one byte of information to another RCX, by infra-red transmission. The value to send, `i`, will be truncated if it can't fit in a byte.

```
int receiveByte ()
```

*Purpose:* Waits for and returns a new byte sent by another RCX via infra-red transmission.

## 14 Math

### 14.1 Purpose

The following are some commonly used numeric constants and functions.

### 14.2 Constants

```
final int MAX_INT
```

*Purpose:* A constant holding the maximum value an `int` can have,  $2^{31} - 1$ .

```
final int MIN_INT
```

*Purpose:* A constant holding the minimum value an `int` can have,  $-2^{31}$ .

```
final double PI
```

*Purpose:* The closest `double` approximation to  $\pi$ .

## 14.3 Methods

`double abs (double a)`

*Purpose:* Returns the absolute value of `a`.

`int abs (int a)`

*Purpose:* Returns the absolute value of `a`.

`double ceil (double a)`

*Purpose:* Returns the least double value that is greater than or equal to `a` and equal to an integer.

`double exp (double x)`

*Purpose:* Returns  $e^x$ , that is Euler's constant  $e$  raised to power  $x$ .

`double floor (double a)`

*Purpose:* Returns the greatest double value that is less than or equal to `a` and equal to an integer.

`double log (double x)`

*Purpose:* Returns the natural logarithm of  $x$ .

`int round (float a)`

*Purpose:* Returns the closest `int` to `a`.

`double sqrt (double a)`

*Purpose:* Returns the square root of `a`.

*Precondition:*  $a \geq 0.0$ .

`double pow (double a, double b)`

*Purpose:* Returns `a` raised to the power `b`,  $a^b$ .

`double sin (double a)`

*Purpose:* Returns the trigonometric sine of `a` radians.

`double cos (double a)`

*Purpose:* Returns the trigonometric cosine of `a` radians.

`double tan (double a)`

*Purpose:* Returns the trigonometric tangent of **a** radians.

```
double asin (double a)
```

*Purpose:* Returns the trigonometric arc sine of **a** in radians.

```
double acos (double a)
```

*Purpose:* Returns the trigonometric arc cosine of **a** in radians.

```
double atan (double a)
```

*Purpose:* Returns the trigonometric arc tangent of **a** in radians.

```
double max (double a, double b)
```

*Purpose:* Returns the greater of **a** and **b**.

```
int max (int a, int b)
```

*Purpose:* Returns the greater of **a** and **b**.

```
double min (double a, double b)
```

*Purpose:* Returns the lesser of **a** and **b**.

```
int min (int a, int b)
```

*Purpose:* Returns the lesser of **a** and **b**.

```
double random ()
```

*Purpose:* Returns a random value  $x$  such that  $0.0 \leq x < 1.0$ .

## 15 Strings

### 15.1 Purpose

The following are methods for working with **Strings**.

### 15.2 Methods

```
int length (String s)
```

*Purpose:* Returns the length of **s**.

```
char charAt (String s, int i)
```

*Purpose:* Returns the character at position **i** in **s**.

*Precondition:*  $0 \leq i < \text{length}(s)$ .

`boolean equals (String a, String b)`

*Purpose:* Returns `true` if and only if `a` contains the same sequence of characters as in `b`.

`boolean parseBoolean (String s)`

*Purpose:* Returns `s` converted to a `boolean`.

`int parseInt (String s)`

*Purpose:* Returns `s` converted to an `int`.

`long parseLong (String s)`

*Purpose:* Returns `s` converted to a `long`.

`float parseFloat (String s)`

*Purpose:* Returns `s` converted to a `float`.

`double parseDouble (String s)`

*Purpose:* Returns `s` converted to a `double`.

# Index

A, 5  
abs, 9  
acos, 10  
asin, 10  
atan, 10

B, 5

C, 5  
ceil, 9  
charAt, 10  
clear, 7  
cos, 9

equals, 11  
exp, 9

floor, 9

getLight, 4  
getRotation, 5

isPushed, 3

lampOff, 6  
lampOn, 6  
length, 10  
LIGHT, 2  
log, 9

main, 2  
max, 10  
MAX\_INT, 8  
min, 10  
MIN\_INT, 8  
motorBackward, 6  
motorFloat, 6  
motorForward, 6  
motorStop, 6

parseBoolean, 11  
parseDouble, 11  
parseFloat, 11  
parseInt, 11

parseLong, 11  
PI, 8  
playTone, 7  
pow, 9  
putChar, 8

random, 10  
receiveByte, 8  
refresh, 8  
resetRotation, 5  
ROTATION, 3  
round, 9

sendByte, 8  
setUpSensor, 3  
showNumber, 7  
sin, 9  
sleep, 7  
sqrt, 9  
systemSound, 7

tan, 9  
TOUCH, 2

waitForDark, 4  
waitForDarker, 4  
waitForLetGo, 3  
waitForLight, 4  
waitForLighter, 4  
waitForPush, 3  
waitForRotation, 5