ROBOTC TUTORIAL PACKET II

Written By: Elizabeth Mabrey

Last update: Dec, 2015

Before you use this document:

Scope: This tutorial will introduce a few advanced topics in data structure, intrinsic RobotC functions.

Special note if you use version RobotC version 4.X+:

In order to make a single version compatible for both NXT and EV3, there have been a few minor changes on the API calls. Take a look into the last chapter in Packet I, and last chapter in this Packet.

Table of Contents

CH1 – INTRO TO PREPROCESSOR STRUCTURE

When a data is declared as macro, it means it will be processed before it is complied.

By convention, **macros** are written in ALL CAPS.

A good candidate for Macros definition: A symbol which contains a value which must not change throughout the entire lifespan of your program.

There are 2 major preprocessing types: macros substitution, and Conditional compilation

1.1 MACROS SUBSTITUTION - #DEFINE

e.g.

Sample code To Declare	- the sample in the body of the code
`#define LM motorA`	`motor[LM] =50;`
`#define LEYE S2`	`X = SensorValue[LEYE];`
`#define seeSilver (SensorValue[S1] > 70)`	`If (seeSilver)` `  nxtDisplayTextLine(2, "bot see silver");`
`#define inRange(hi,lo,x) (x<=hi && x>=lo)`	`If ( inRange(90, 10, data)` `  nxtDisplayTextLine(2, "10<=%d<=90", x);`

Side note: Although RobotC is a C-based programming language, it is not 100% ANSI-C compliant.

ANSI-C is the standard published by the American National Standards Institute (ANSI) for the C programming language in 1989 to encourage portability. C programming language is the most ubiquitous high level programming language closest to the machine level before assembler language. In low-level development system like robotics system, or embedded system, most use C along with assembler.

Since Robotc is not 100% ANSI-C compliant. There are some intrinsic functions may not conform to the standard signature, or implementation.

1.2 HEADER FILES DIRECTIVES - #INCLUDE

The #include directive tells the preprocessor to include another file, and place it directly into the current file. You should place #include directives at the top of a program.

e.g.

#include	`#include "myLibrary.c"`	`// myLibrary.c contains your own collection of codes`

1.3 CONDITIONAL COMPILATION DIRECTIVES - #IFDEF … #ELSEIF … #ENDIF

#ifdef
…
#endif
	`#ifdef doLog`
	`  performDataLog();`
	`#endif`
		`Function "performDataLog()" will be called if macros "doLog" has been defined`

#undef	`#undef doLog`	`// to un-defined a macro named "doLog"`

1.4 LEARN FROM SAMPLES

p0preProcessor.c

p1preProcessor.c

CH2 – INTRO TO DATA ABSTRACTION

2.1 TYPEDEF

typedef

```
typedef    float    AMOUNT;
…
       AMOUNT  balance;
balance = 3.49;
```

2.2. ENUMERATION

Let say you want to use a macro to represent a certain set of valid values.

e.g.:

```
#define NORTH        0
#define SOUTH        1
#define EAST         2
#define WEST         3
```

Instead, you create a type to ensure there are only valid values of the same kind:

typedef enumWord { NORTH, EAST, SOUTH, WEST } DIR;

There are only 4 valid values are allowed for type "DIR" : NORTH, EAST, SOUTH, WEST or 0, 1, 2, 3. However, anything else is invalid.

typedef enumWord

```
typedef enumWord {  NORTH, EAST, SOUTH, WEST } DIR;
…
   DIR mydir;
   mydir = NORTH;
   mydir = EAST;
```

```
   typedef enumWord
   {
     AK, AR, NJ, NY
   } STATE;
   …

   STATE myState;
   …
   myState = NJ;
```

2.3 TYPEDEF STRUCT

typedef struct

```
typedef struct{
    int yob;
    char gender;
    string fullName;
} STUDENT;

…
STUDENT you;
…
you.yob = 1999;
you.gender = 'F';
you.fullName = "Joe Smith";
```

2.4 LEARN FROM SAMPLES

simpleStruct.c

p0simpleEnum.c

p1simpleEnum.c

p2simpleStruct.c

CH3 – MORE ROBOTC INTRINSICS

3.1 BASIC MATH FUNCTIONS

You can obtain the list from the "Function Library".

To access the description, you need to activate the "Functions Explorer" under "View".

Some of the most commonly used:

```
abs(input)
sqrt(input)

abs(float n)
exp(float n)
sqrt(float n)

srand(int seed)
range(int range)
```

Basic trigonometry functions.

```
float sin(fRadians)
float cos(fRadians)
float sinDegrees(degrees)
float cosDegrees(degrees)
```

PI : is a macro

3.2 TIMER CONTROL

You have used the simple timing functions **wait1Sec(int millisecond)**, and **wait10Sec(int 10s milliseconds).** However, the program will have to wait the specified time elapsed before it can go onto the next instruction.

RobotC provides 4 individual system timers accessible by users, **T1, T2, T3 and T4**. This will allow your program to proceed to the next instruction.

Look at this almost like the timer watch we use. You can reset to 0 in programs, and then used to monitor elapsed time and control program flow.

RobotC provides 3 array data types to retrieve the value.

time1[T1]: retrieves the value of timer T1 in units of 1-msec
time10[T1]: retrieves the value of timer T1 in units of 10-msec
time100[T1]: retrieves the value of timer T1 in units of 100-msec

Samples:

```
ClearTimer(T1);
while (time1[T1] < 10000)  // robot will go forward
{     motor[motorA] = 50;
      motor[motoB] = 50;
      nxtDisplayTextLine(5, "Elaped time: %d", time[T1]/1000);
}
```

Note: The value returned is a signed integer, so each array will meet its upper bounds at a value of 32,768 ticks.

3.3 MULTITASKING

A task is a module which runs simultaneously with other tasks.

To run **additional tasks** simultaneously:

StartTask (TaskID);

You may stop it by:

StopTask(TaskID);

e.g.

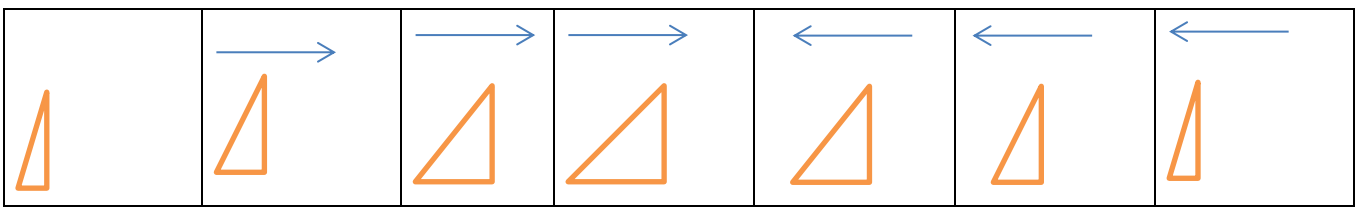
```
task  playMusic()
{
    while (1)
    {
          … do something  …
    }
}

task main()
{
  StartTask(playMusic);

  ClearTimer(T1);
  while (time1[T1] < 10000)
  { motor[motorA] = 50;
    motor[motorB] = 50;
    nxtDisplayTextLine(5, "Elaped time: %d", time1[T1]/1000);
  }
}
```

3.4 LEARN FROM SAMPLES

p0displayScroll.c p1Button.c p2timerSample.c p4multiTask.c

3.5 MINI-CHALLNGE EXERCISES

Modify simpleTrigWButtons.c so that the angle should always be >0 & <=89. Use the simpleTrigWButtons.c to draw the triangle on the screen. For example:

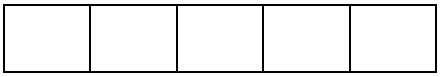


CH4 STRING/ARRAY MANIPULATION

4.1 ARRAY

In computing, an array means a collection of memory fields with the same data types and arranged in a contiguous sequence manner.

e.g. int x; // a single variable integer represented by symbol x in the memory.
int x[10]; // 10 integer values represented by symbol x[0], x[1], x[2],…. [9] in the memory.
Note that their memory location is contiguous to each other.



X[0] X[1] X[2] … X[9] Note that it is from [0] to [9] . x[10] is out of bound

NOTE: any data type can be expressed in an array. In this segment, we will cover 2 types data array only.

e.g.

```
char cVar1[50];            // User must explicitly declare size. The dimension 50 is arbitrary.
byte iVar[25];
string sVar[10];
STUDENT one[200];      // based on the new data type "STUDENT" created in earlier chapter
```

Samples for assigning values to the array:

```
cVar[4] = 'k';
iVar[3] = 10;
sVar[9] =  "north";
one[100].gender = 'M';
```

Invalid:

```
cVar[4] = "abc";  // invalid value
iVar[25] = 10;     // out of bound
sVar[-1] =  "north";    // out of bound
one[100] = 'M';  //  invalid syntax
```

4.2 BASIC STRING OPERATION

Word of caution to the experienced C programmer:

RobotC is not fully ANSI-C compliant. They have a very customized way to deal with String. You cannot treat String as a memory pointer.

Therefore, you cannot map address between a string and an array structure.

Date Type: *string*

A 'string' is an array of maximum 19 characters terminated with a 'null' (zero value) character. In the ideal solution, you'd be able to manipulate strings just like numeric variables as shown in the following examples.

e.g.

```
string str1 = "ROBOT";     // declare a string variable named str1 and
                           // initialize it with "ROBOT".
string str2 = "C";         // 'str2' contains "C" now
string str3;               // declare str3, with no data

str3 = str1+str2;          // str3 now contains "ROBOTC"
```

special note: It imposes maximum 19 bytes, not like most C compliant software which does not impose such constraint, but as much as memory will allow.

Common functions:

(again, not all functions are ANSI-C compliant!)

- *int strcmp(str1, str2)*
  - return 0 if str1 == str2
  - return >0 if str1 > str2
  - return <0 if str1 < str2
- void strcat(str1, str2)
  - append value of str2 to the end of str1.
  - str1 will contain "ROBOTC"
- void strcpy(str1, str2)
  - copy str2 into str1.
  - str1 will contain "C"
- void memcpy(str2, str1, n)
  - works like strcpy(), except this is a safer version because you can determine only "n" bytes will be copies.
  - So `memcpy(str2, str1, 3)` will result str2 contains "ROB".
  - You must "null-terminate" str2 yourself. In a ANSI-C compliant compiler, you can use simple "pointer math" to locate the end of the "str2" to put the "null" in. However, in RobotC, you can `memset(str2, 0, numOfBytes)` before `memcpy(…)` is called.

4.3 String Parsing

This section will show you how to parse a string.

For example:

```
string fullstr="2;0.145;xyz;3;9.12;";
const string delimitor = ";";
int idx=0, line=0;

idx = StringFind(fullstr, delimitor);
if  (idx>-1)
{
    memset(sdest, 0, 19);
    memcpy(sdest, fullstr, idx);
    nxtDisplayTextLine(line, "%7s", sdest);
}
StringDelete(fullstr, 0,idx+1);  // now, fullstr = "0.145;xyz;3;9.12;"
…
```

4.4 Pointer

- A pointer is a variable whose value is the address of another variable.
- If used in parameters passing between functions, it is called called pass-by-reference. Pass by reference refers to a method of passing the address of an argument in the calling function to a corresponding parameter in the called function.

See sample : ** *p7Pointer with string.c* ** *p6Pointer w num.c* ** *p8PointerAdv.c*

4.5 Learn from Samples

P0array.c	p3string.c	** *p6Pointer w num.c*
p0string.c	** *p4stringBytes.c*	** *p8PointerAdv.c*
p1string.c	** *p5Ascii.c*	
p2string.c	** *p7Pointer with string.c*	

Fix the following programs:

errPointer01.c	errString4.c	errString7.c
errString1.c	errString5.c	
errString2.c	errString6.c	

4.6 Exercise

1. Complete simpleBytesArrayParsing.c.
   - This program has initialized a byte array fullstr[19] with "123.5687".
   - You need to write a function called atof(…) to convert the byte array to a float number.
   - Your function's signature (prototype) should look like this:

```
float atof( ARR a)
```

```
where ARR is a new struct data type:
     typedef struct
{
}  ARR;

   ubyte arr[19];
```

2. Write a program to parse the string "42.999,N,78.7824,W". Your program will display:

Latitude = 42.999 Longitude = 78.7824

CH5 - WORKING WITH BINARY

5.1 ABOUT THE BASICS

Know about the Base:

Based 10 (decimal): 234_{10} == $2 * \mathbf{10^2} + 3 * \mathbf{10^1} + 4 * \mathbf{10^0}$

Based 16 (hexadecimal): 150_{16} == $1 * \mathbf{16^2} + 5 * \mathbf{16^1} + 0 * \mathbf{16^0}$

Based 2 (binary) : 101_2 == $1 * \mathbf{2^2} + 0 * \mathbf{2^1} + 1 * \mathbf{2^0}$

All numbers are stored in binary in the computer. What you use is just a form of representation to human for various application.

Decimal	Hex	Binary	Decimal	Hex	Binary	Decimal	Hex	Binary
0	0	0	16	10	10000	143	8F	10001111
1	1	1	17	11	10001	246	F6	11110110
2	2	10	18	12	10010	2184	888	100010001000
3	3	11	19	13	10011	3848	F08	111100001000
4	4	100	20	14	10100			
5	5	101	21	15	10101			
6	6	110	22	16	10110			
7	7	111	23	17	10111			
8	8	1000	24	18	11000	Convert the following hex to both Decimal and Binary based...		
9	9	1001	25	19	11001	On your own, no calculator		
10	A	1010	26	1A	11010		F00	
11	B	1011	27	1B	11011		11C	
12	C	1100	28	1C	11100		C0C	
13	D	1101	29	1D	11101		E00	
14	E	1110	30	1E	11110		B00	
15	F	1111	31	1F	11111		A00	

In RobotC, base representation is done as following:

```
int x =   29;         // base 10
int x =  0x1D;        // base 16
int x =  0b11101;     // base 2
```

Bit-AND / Bit-OR operations

		Bit 3	Bit 2	Bit 1	Bit 0
	X	1	1	0	0
	Y	1	0	1	0
Bit-And	Z = X & Y	1	0	0	0
Bit-OR (inclusive)	Z = X | Y	1	1	1	0
Bit-XOR (exclusive)	Z = X ^ Y	0	1	1	0
Bit-NOT	Z = ~X	0	0	1	1

BitShift Operation

x **<<** *n* means the value x is shift left for "n" number of bits

x **>>** *n* means the value x is shift right for "n" number of bits

left shift for N bits == 2^N Right shift for N bits == $1/2^N$ 2^{-N}

Caution! Do not use bit shift unless you FULLY aware of the signed and unsigned relationship, as well as binary math operation. If you have to use it, you should write a separate program to test out all signed and signed situation to ensure you will not have shifting errors.

Practical Example

Example 1: Say you use 3 touch sensors as your remote controller to control driving a car.

State Table for this RC algorithm for a bot with 2 motors

T# where T= Touch sensor, # = input port number
T1 : control Left motor only
T2 : serves as logic state switch for different set of motions
T3 : control Right motor only

Action		T1	T2	T3
X = stop ->= forward <- = backward		0 = off/release 1 = on/press		
1	All motors X	0	0	0
2	Right-> Left X; i.e. left drag turn	0	0	1
3	Left-> Right X; i.e. right drag turn	1	0	0
4	Both-> ; i.e. forward	1	0	1
5	Both<- ; i.e. backup	0	1	0
6	Right-> Left<-; i.e. left pivot turn	0	1	1
7	Left-> Right<-; i.e. right pivot turn	1	1	0
8	Anything else you choose.	1	1	1

To utilize the bits operation to identify the corresponding motions:

```
#define T3ONLY  1
#define T2ONLY  1<<1
#define T1ONLY  1<<2
#define T1T2    T1ONLY | T2ONLY
#define T1T3    T1ONLY | T3ONLY
#define T2T3    T2ONLY | T3ONLY
#define NONE    0
#define ALL     T1ONLY | T2ONLY | T3ONLY
...
    result =  SensorValue[S1]<<2  |   SensorValue[S2]<<1  |
              SensorValue[S3];
...
    switch (result)
    {
     case T2ONLY:
        …; break;
     …
  case ALL :
     …; break;
```

Example 2: Say you use 3 light sensors to handle line tracing with gap.

```
#define L3ONLY  1       //light sensor port 3 sees dark
#define L2ONLY  1<<1    //light sensor port 2 sees dark
#define L1ONLY  1<<2    //light sensor port 1 sees dark
#define L1L2    L1ONLY | L2ONLY
#define L1L3    L1ONLY | L3ONLY
#define L2L3    L2ONLY | L3ONLY
#define NONE    0
#define ALL     L1ONLY | L2ONLY | L3ONLY
int light1Threshhold = 50;
int light2Threshhold = 45;
int light3Threshhold = 51;
...
int result =(SensorValue[S1]<=light1Threshold) <<2  |
            (SensorValue[S2]<=light2Threshold) << 1 |
             SensorValue[S3]<=light3Threshold) ;
...
    switch (result)
    {
    case L1ONLY:
     …
    case ALL :
     …; break;
     ...;
```

5.2 LEARN FROM SAMPLES

bytesbits01.c
p0bytesbits.c
p1bytesbits.c
p3bytesbits.c
p4bytesbits.c

5.3 EXERCISES

Ex.1) Write one or two functions which will display the number in binary. The number can be one byte "char" type or "int" type. For example:

For char x=0xc6, your display should show 11000110

For int x=0xc6, your display should show 0000000011000110

Ex.2) Write 4 function to perform +, -, *, / operations without using the operator itself. Yes, only using bit-wise operations.

CH6 – DATA LOGGING

Data logging is an important concept for you understand any type measurement over time. In your case, you most likely will record sensor feedback over a period of time. This allows you to:

- Perform sensor data analysis.
- Find any possible anomaly during run time.
- Diagnose errors.

For some platforms, you may use a physical datalogger device. In Mindstorms, you will need to perform data logging yourself using file accessing method.

6.1 FILE ACCESS

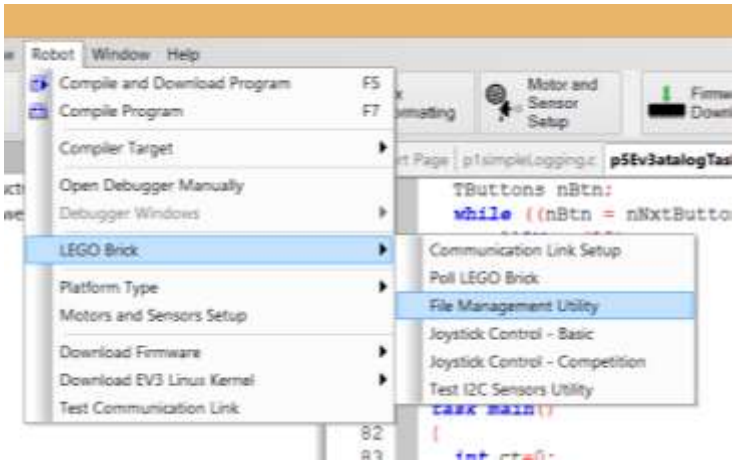
Typically, steps to follow:

1. Determine what sensor data you want to collect.
2. Determine the frequency for probing
3. Determine minimum and maximum data points.
4. Create a file name, e.g. sprintf(str, "log%d.txt", nSysTime()/1000)
5. Create a the file, i.e open the new file
6. Probe the device which you are interested to collect feedback from for a period of time
7. Write the data to the file
8. Close file
9. Download the file to the computer and use some tool like Excel to generate report or draft graph.
10. Important: you should generate multiple log files with the identical set of control variables in order to maximize the accuracy.

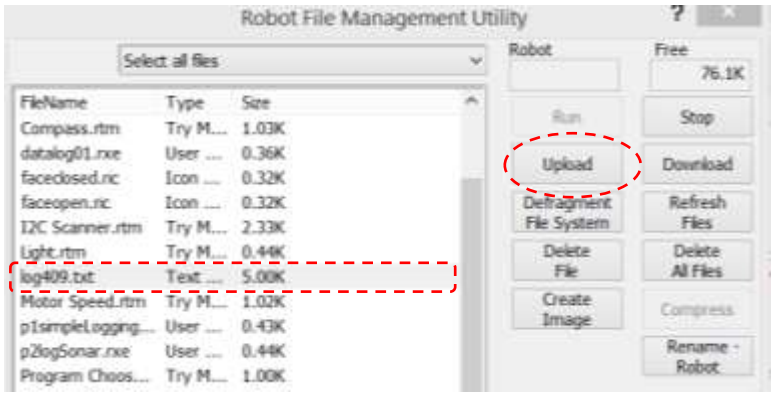
The next section, 6.2, will show you how to upload the log files to your computer for further analysis. You should do that for each sample you learned to write in section 6.3.

Be resourceful. You may look up all the available RobotC Instrinsic File Access functions using the Help feature.

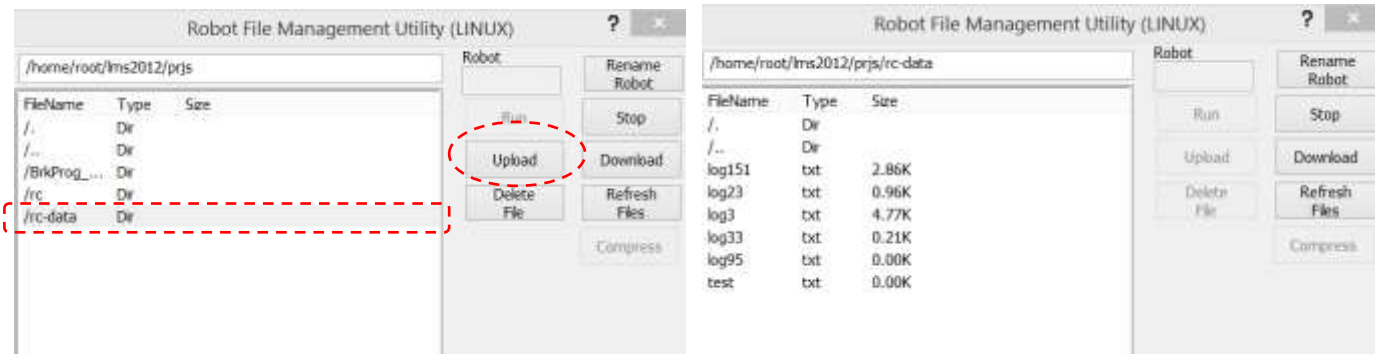
6.2 VIEW THE LOGGED FILE



NXT File Management Utility



EV3 File Management Utility



After you upload the selected log file(s), you may load it into the spreadsheet and generate report or graph to help you understand how

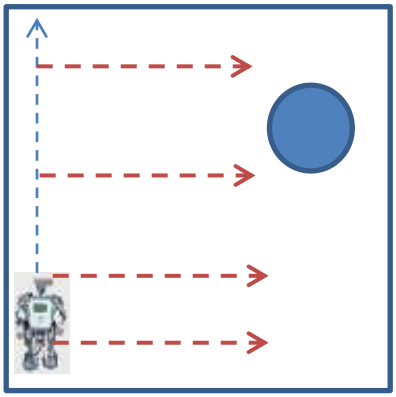
6.3 Learn from Samples

- p1simpleLogging.c : simple logging with counter and timer.
- p2simpleLogging02.c : log the distance feedback from ultrasonic sensor. This exercise requires to use a UltraSonic sensor.
- p3logButtonPress.c : log the time when a button is pushed. Data is logged into a single file.
- P4logButtonPress.c : same as p3logButtonPress.c, except it allows variable log file name
- p5datalogTask.c : datalogging is done as a background task to run simultaneously with the main process.

6.4 Exercises

The following two exercises require your robot to have following configuration:

- #define frontEye S1 // ultrasonic sensor
- #define sideEye S3 // ultrasonic sensor
- #define LM motorA#define RM motorC



Ex1) Write program to log the ultrasonic value(s) while the robot runs for 90 cm straight. Probe and log the sensors value for at least 100 points (you will need to decide the interval value).

Each Data record should contain:

Encoder value ; sideEye value; FrontEye value

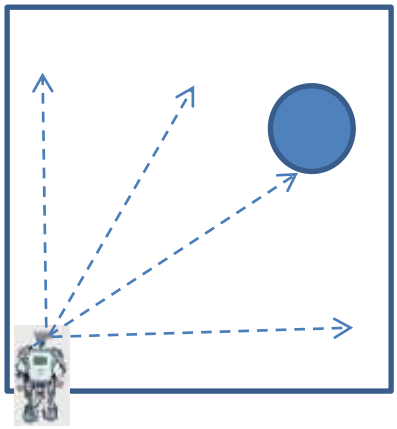
(Distance for side Eye - - ->)

(Distance for front Eye - - ->)

Ex.2) Your robot needs to pivot on a point. It needs to log the distance while it pivots for 90 degrees. It needs to probe and log the sensor value at least once for each degree change. Therefore, you should have at least 90 data points.

Each Data record should contain:

Turning degrees; FrontEye value



CHAPTER 7- FLOAT NUMBER MANIPULATION

You will only need a NXT to do complete this phase.

Your program should do the following:

- Create two strings
  - string token= "42.999,N,78.7824,W"
  - string token= "42.912,N,78.7601,W"
- Parse them into their own x & y:

  Point 1: x=78.7824 y=42.999

  Point 2: y=78.7601 x=42.912

  Where x, y are in the unit of meters.

- Draw a line on the LCD screen from point 1 as your origin, and point 2 as the destination, with an arrow
- Display the angle delta of orientation from point 1 to point 2
- Display the distance from point 1 to point 2

CHAPTER 8 SOME DIFFERENCES IN EV3

This chapter covers some commonly used APIs which are different in EV3 from NXT.

Some minor changes in data types changed, such as from "word" to "short".

You should always consult either the RobotC Helper and/or the "right mouse click" on the API itself to make sure the data types matches.

Since these are just small changes, I do not re-run all samples for version 4.X. You should have enough skill set to make these slight changes yourself.

List of changes on the topics covered in this packet.

1) File Access / Data logging:

- Simple open:

```
string sFileName = "…file name..";
int N = …some number for # of bytes to write …;
NXT:
    TFileHandle fp;
    TFileIOResult result;
    OpenWrite(fp, result, sFileName, N);

EV3:
    long fp;
    fp = fileOpenWrite(sFileName);
```

- Simple write:

```
string str = "…something…";
NXT:
         WriteText(fp, result, str);
EV3:
         fileWriteData(fp, str, strlen(str));
```

- close the file:

```
NXT:
         Close(fp, result);
EV3:
         fileClose(fp);
```

To learn more, may download the samples code from Packet II – Chapter 6 samples.

Be resourceful, use the "Help", or RobotC samples, or look into the EV3RobotIntrinsics.h.

2) Color-RGB sensor access:

```
NXT:
      short rgb[4];
      SensorType[S1] = sensorColorNxtFULL;
      getColorSensorData(S1, colorRaw, rgb);

EV3:
      long r, g, b;
      SensorMode[S1] =   modeEV3Color_RGB_Raw;
      getColorRGB(S1, r, g, b); // note: it uses the C++ compliant pointer reference syntax
```

***** About Bluetooth :**

There are yet complete Bluetooth implementation in EV3 yet. For NXT, access it from http://learning.stormingrobots.com

The End of Packet II