RoboCup Rescue

EV3 Workshop

Part 1

Introduction to RobotC

Why use RobotC?

- RobotC is a more traditional text based programming language
- The more compact coding editor allows for large programs to be easily developed, debugged and managed
- RobotC also has some programming advantages over the EV3-Graphical software, such as;
  - Re-usable Functions
  - Complexity of Algorithms
  - Complexity of Calculations
  - Smaller more efficient running of code on the robot
  - A more traditional code layout that allows for easy pseudocode translation

Note: BricxCC/NXC does not at this time support EV3. It does support NXT and is a 'free to use' alternative

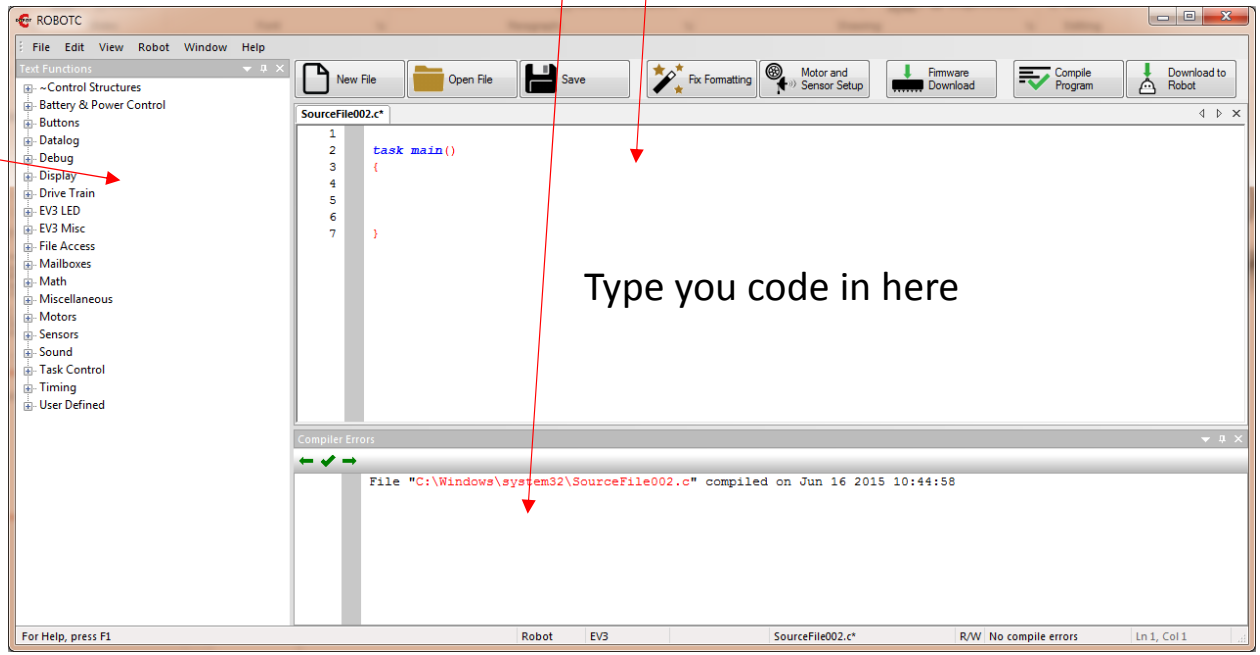
RobotC and RoboCup Rescue

- Should be considered by some Secondary and Open teams
- Lego's EV3 software can be used for complex algorithms, however, it becomes extremely difficult to visualise the entire program
- RobotC allows teams to develop more complex and precise solutions
- Teams can develop reusable Functions (a little like MyBlocks)
- Tasks are easier to manage (turn on and off)

- Teams need to realise that there is an initial steep learning curve!
- Prior procedural programming experience is an advantage!

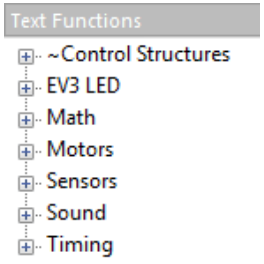
The IDE (Integrated Development Environment)

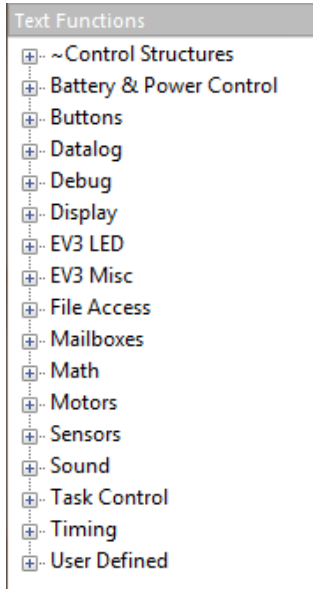
- RobotC 4.xxx has a traditional text editor code window including line numbering and colour coded text
- A basic Debug window displays compilation errors
- A Text Functions list which I would rather call the Control Structures and Keyword Templates window
- RobotC does include some IntelliSense code completion delivering logical code in a selectable dropdown list

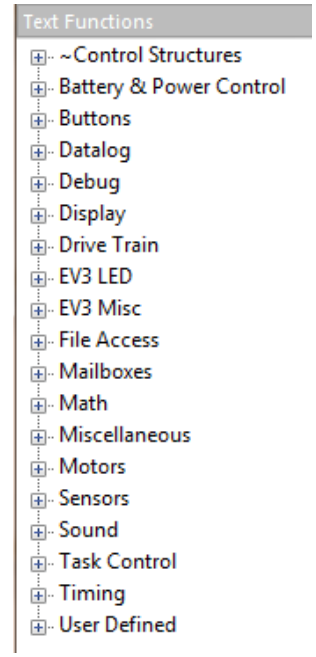


Text Functions pallet

- It is impossible for someone to remember ALL of the code syntax for a particular language!
- The Text Functions pallet contains all the available RobotC Functions
  - used to control the actuators (motors etc) and
  - receive information from all the sensors and finally
  - process all the inputs using control structures to allow the robot to navigate the course.
- Three levels of menus are available
  - Basic, Expert & Super User
  - Change this from the Windows menu







Programming Syntax and Rules

- Here is an example program – a Left Edge Line Follower
  - Each program must have an Entry Point
  - A Task must have its own pair of { }
  - All Functions have their own pair of ( )
  - Each Control Structure (While, If .etc) has its own pair of { } between which your process code is placed
  - Every Expression must end with a ; except for Boolean Expressions
  - [ ] brackets are used with Variable functions
  - // are used to write a comment

```
1  #pragma config(Sensor, S1, LS1,      sensorEV3_Color)
2  #pragma config(Sensor, S4, LS2,      sensorEV3_Color)
3  //*!!Code generated by 'ROBOTC' config wizard  !!*//
4
5  task main()
6  {
7    while (true)
8    {
9      if (getColorReflected(LS1) > 50)
10     {
11       setMotorSpeed(motorB, 25);
12       setMotorSpeed(motorC, 0);
13     }
14     else
15     {
16       setMotorSpeed(motorB, 0);
17       setMotorSpeed(motorC, 25);
18     }
19   }
20 }
```

Tasks - Multitasking

```
task TOne()
{
  while(true)
  {
    wait1Msec(300);                              // Allow for a short wait, freeing up the CPU for other tasks.
    displayCenteredBigTextLine(3, "TASK 1");     // Display that TOne is running simultaneously with Main.
    displayClearTextLine(6);                     /* Erase TTwo's display since it isn't doing anything.              */
    displayClearTextLine(7);                     /* (Even though technically it is still running in the background.) */
  }
  return;
}
//=============================================================================================================================
task TTwo()
{
  while(true)
  {
    wait1Msec(300);                                 // Allow for a short wait, freeing up the CPU for other tasks.
    while(SensorValue(S1) == 1)            // While Touch Sensor is activated:
    {
      stopTask(TOne);                               // Kill TOne.
      displayCenteredBigTextLine(6, "TASK 2");      // Display that TTwo is running simultaneously with Main.
      displayClearTextLine(3);                      /* Erase TOne's display since it was killed.                          */
      displayClearTextLine(4);                      /* (TOne is actually gone for now so only TTwo and Main are running.) */
      wait1Msec(100);                               // Wait a small amount to allow the NXT Display to display correctly.
    }
    startTask(TOne);                                // Start TOne again, running all 3 tasks simultaneously.
  }
  return;
}
//=============================================================================================================================
task main()
{
  startTask(TOne);                          // Start Task TOne.
  startTask(TTwo);                          // Start Task TTwo.
  while(true)
  {
    wait1Msec(300);                             // Allow for a short wait, freeing up the CPU for other tasks.
    displayCenteredBigTextLine(0, "TASK M");     // Display that Main is running.
  }
  return;
}
```

- Tasks are activated in task main() when the program starts
- Task main() is always at the bottom and always running
- All other tasks() are positioned above task main()
- All tasks can start and stop other tasks (except for task main(), it's always running and can't be stopped by another task

Variables – Global vs Local

- In Mindstorms EV3, all variables are Global
- RobotC allows for Global and Local variables
- Consider the example code which has 1 global, and 2 local variables
- Whenever the global variable is called it will display the global value
- The local variables in this example have the same identifier k but reside in separate control structures as defined by { }
- This results in 'k' displaying different values (25 & 10) depending on its location

```
1  int i = 5;  //Global Variable
2  task main()
3  {
4    int k = 10; //Local Variable
5    {
6      int k = 25; //Local Variable
7      displayTextLine(1, "%d", k); // this will display 25
8      displayTextLine(1, "%d", i); // this will display 5
9    }
10   displayTextLine(1, "%d", k);   // this will display 10
11 }
```

Data Types

- Variables need to be declared as a particular Data Type.

Type	Syntax	Description
• Boolean:	bool name;	has only two values – True or False
• Integer:	int name;	holds 2 byte integer -32768 to 32767
• Long Integer:	long name;	holds 4 byte integer +- 2.1 million
• Floating Point:	float name;	same range as int, are decimal/fractions
• String:	string name;	can hold a sequence of characters

Unary Operations – Lazy programmers

- C is full of shortcuts because programmers are inherently lazy
- Rather than writing i = i + 1 for an accumulator, we can write ++i
- Here are examples of code shortcuts or Unary Operations
- ++i is the same as i = i + 1 which is the same as i +=1
- --i is the same as i = i - 1 which is the same as i -=1
- i /=2; is the same as i = i / 2
- i *=2; is the same as i = i * 2
- i %=2; is the same as i = i % 2

Functions

- Is a group of statements that run as a single unit when it is called from another location such as task main( )
- If you find that you are re-typing the same code to do the same action, then it's time to create a Function
- Firstly, Declare your Function by using the word 'void'
- Give the Function a meaningful name (no spaces)
- Write you code between curly braces { }
- Call you function, by name plus brackets and semicolon
- Parameters can be passed within the brackets ( ) allowing the Function to behave differently depending on variable inputs

```
1   void moveForward()
2   {
3     motor[motorB]=50;
4     motor[motorC]=50;
5     wait1Msec(1000);
6   }
7
8   task main()
9   {
10  moveForward();
11  }
```

Control Structures

- Repetition (Loop) is handled by
  - WHILE (conditional expression) {} *-Pre-test loop*
  - DO { *body* } WHILE (conditional expression) *-Post-test loop*
  - FOR (initial setup; conditional expression; increment or decrement) {} *–Counting loop*

- Decision is handled by
  - IF (conditional expression) {body} *-Binary decision*
  - IF (conditional expression) { *body* } ELSE { *body* } *-Binary decision with option*
  - *Nested IF's can create complex decision making structures*
  - SWITCH (variable) { CASE num: *body* break; default; *body* } *-Multiway selection*

Debugger

- Allows you to more easily find Logical Errors in code
- Start and Stop; the program from the PC
- Single Step Through; of code line by line
- Breakpoints; define points in the code to stop execution and view
- Read and Write values of Variables in your program
- Read and Write values of Motors and Sensors

- Another method of debugging robots, is to use dataLogging, allowing you to record in real time all of the variables, motor & sensor values and then display them on a chart or graph for further alalysis