PROJECT LEAD THE WAY

PLTW

Igniting imagination and innovation through learning.

Programming Design

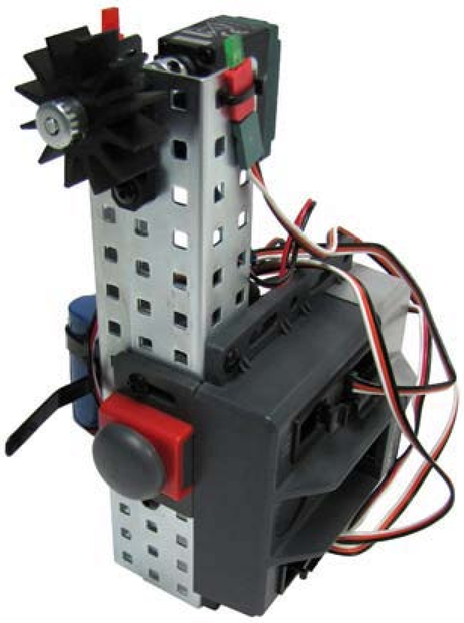
ROBOTC Software

Behavior-Based Programming

- A behavior is anything your robot does
  - Turning on a single motor or servo
- Three main types of behaviors
  1. Complex behaviors – Robot performs a complex task (automated fan control)
  2. Simple behaviors – Simple task performed by the robot (fan stops when sensor activated)
  3. Basic behaviors – Single commands to the robot (turn on a motor)
- Complex behaviors can always be broken down into simple behaviors, which are then broken down into basic behaviors

Complex Behaviors

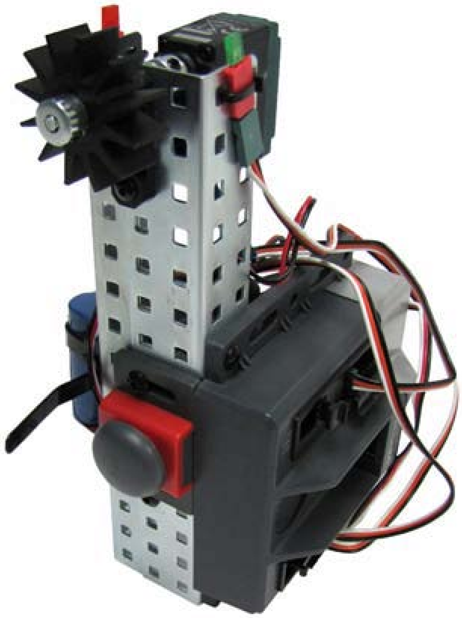
- Describe the task or overall goal that your program will accomplish.
  - A fan will run until someone needs it to stop. A safety device warning light will come on before the fan turns on. Another light will indicate that the fan has stopped.
- This may be described as one or more complex behaviors.



Creating Pseudocode

- Break down your behaviors into individual actions.
- Do not worry about syntax or which commands will be used with ROBOTC.
- Simply describe them in short phrases.
- Example
  - Turn a motor on for three seconds
  - Follow a line until running into a wall.

Simple Behaviors



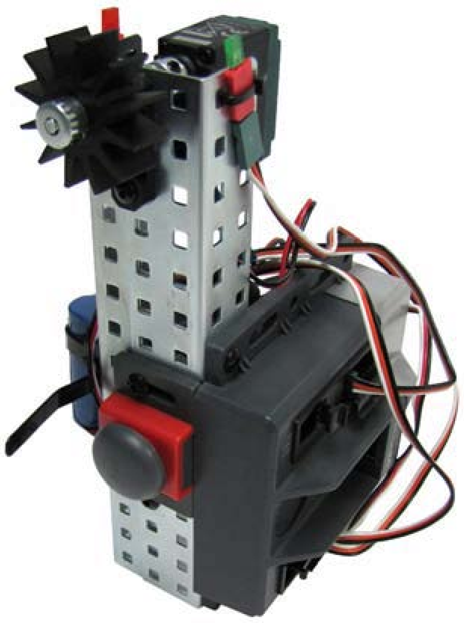
- Break each complex behavior down into simple behaviors.
- List the behaviors line by line in the order that each should occur.
- Describe actions and what prompts each action to continue.

Creating Pseudocode

- Example
  - Warning light comes on before the fan starts for three seconds
  - Fan turns on and runs until a button is pressed
  - A different light comes on for three seconds before the program stops

Basic Behaviors

- Break each simple behavior down further into basic behaviors.
- Think in terms of what each input and output component will be on your device.

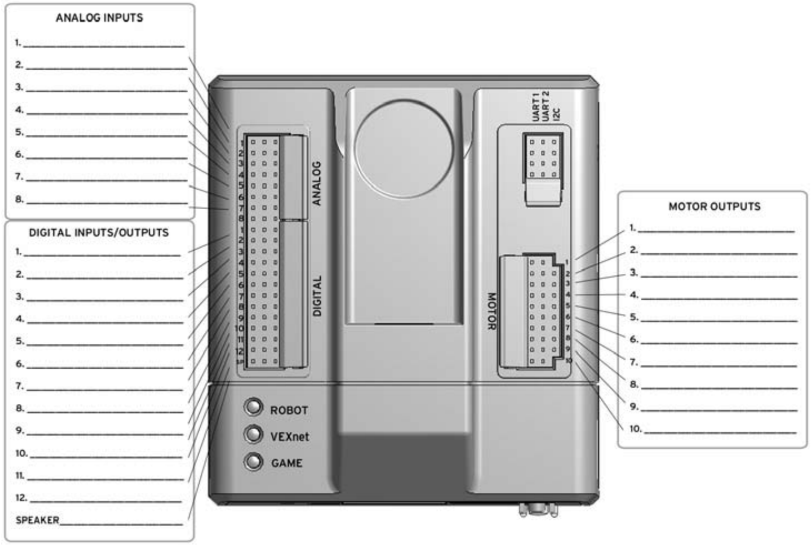


Creating Pseudocode

- Example
  - Program begins
  - Light 1 (LED 1) turns on for three seconds
  - Fan (Motor 1) turns on until a button (bumper switch) is pressed
  - Light 2 (LED 2) turns on for 3 seconds
  - Program ends

Identify Inputs and Outputs

- Identify which ports each input and output will be plugged into on the Cortex
- Pay attention to which sensors are analog and which are digital
- Label planning diagram



PLTW ROBOTC Program Template

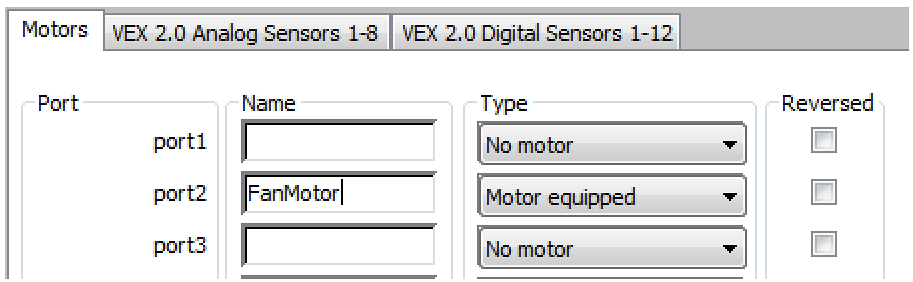
- Open Sample Program PLTWTemplate
- Use the initial description (complex behaviors) of your overall program goal as the task description
- Copy pseudocode (basic behaviors) for the Pseudocode section of the PLTW ROBOTC program template

PLTW ROBOTC Program Template

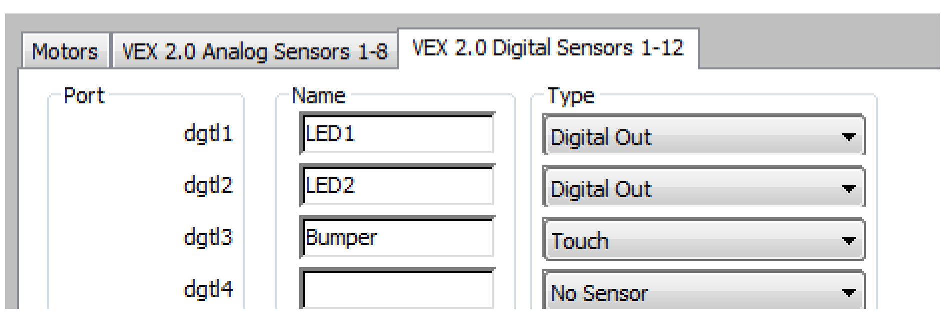
```
/*
  Project Title:
  Team Members:
  Date:
  Section:


  Task Description:

A fan will run until someone needs it to stop. There will be a warning light
as a safety device before the fan turns on and another light to indicate that the
fan has stopped.

  Pseudocode:

Program begins
Light 1 (LED 1) turns on
for three seconds
Fan (Motor 1) turns on
Until a button (bumper switch)is pressed
Light 2 (LED 2) turns on
for 3 seconds
Program ends

*/

task main()
{
 //Program begins
 //Light 1 (LED 1) turns on
 //for three seconds
 //Fan (Motor 1) turns on
 //Until a button (bumper switch)is pressed
 //Light 2 (LED 2) turns on
 //for 3 seconds
 //Program ends
}
```

Program Design

- Identify all inputs and outputs in the Motors and Sensors Setup window.



- Use the Debugger to confirm that all inputs and outputs are working as expected.



Motor 2 for 5 Seconds

Testbed.c | Motor 2 for 5 seconds.c*

```
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard

/*
  Project Title:
  Team Members:
  Date:
  Section:


  Task Description:


  Pseudocode:

*/

task main()
{                                       //Program begins, insert code within curl

  startMotor(rightMotor, 63);
  wait(5.0);
  stopMotor(rightMotor);

}
```

Motor 2 for 5 Seconds

```
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard
```

Displays configuration changes from the Motors and Sensors Setup

```
task main()
{

  startMotor(rightMotor, 63);
  wait(5.0);
  stopMotor(rightMotor);

}
```

Defines the "main task" of the robot

All commands belonging to task main must be in-between these curly braces

Motor 2 for 5 Seconds

```
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard
```

```
task main()
{

  startMotor(rightMotor, 63);
  wait(5.0);
  stopMotor(rightMotor);

}
```

Turns the port2 rightMotor on at half power forward

Motor 2 for 5 Seconds

```
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard

task main()
{

  startMotor(rightMotor, 63);
  wait(5.0);
  stopMotor(rightMotor);

}
```

Causes the robot to wait here in the program for 5.0 seconds

Motor 2 for 5 Seconds

```
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard
```

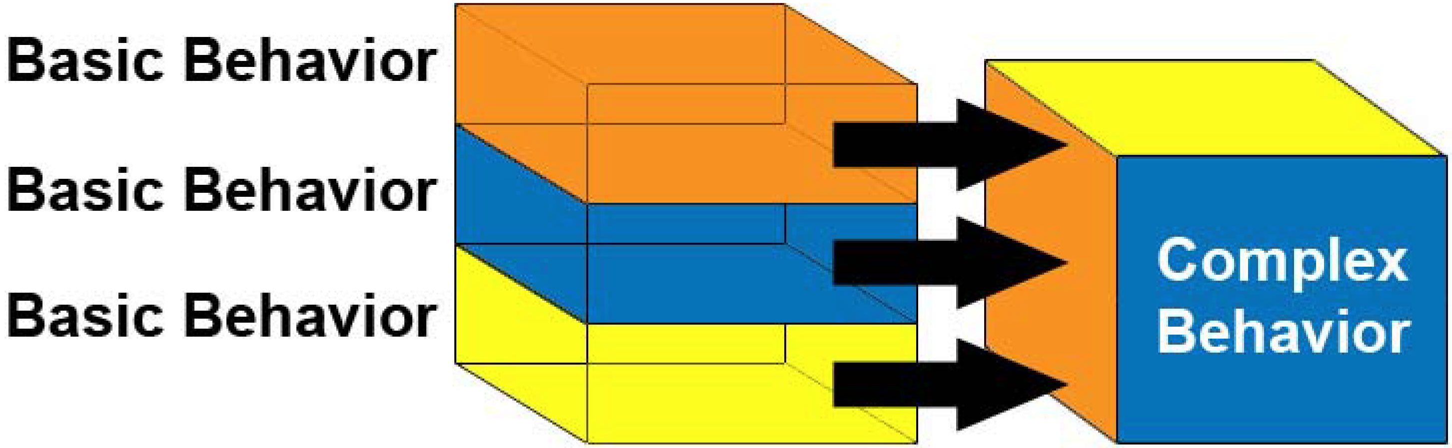
```
task main()
{

  startMotor(rightMotor, 63);
  wait(5.0);
  stopMotor(rightMotor);

}
```

Stops the port2 rightMotor.

End Result: rightMotor spins for 5.0 seconds

Program Design

- Many basic behaviors generally come together to create a complex behavior.
- Troubleshoot basic behaviors as they come together to form a complex behavior.



Program Design

- Code and test small behaviors or sets of behaviors individually.
- Edit or add comments as you build code.

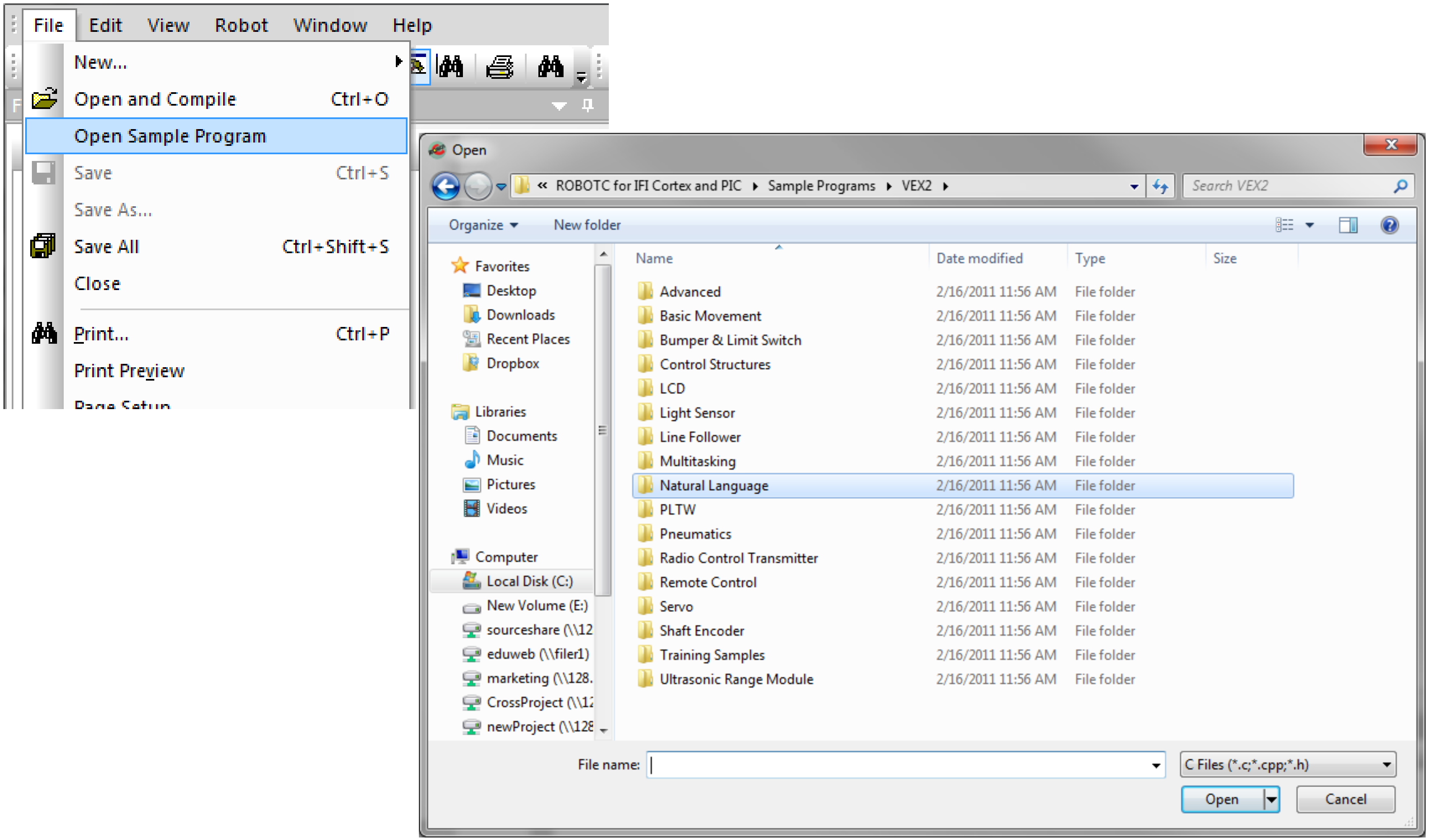
```
task main()
{
                                    //Program begins
turnLEDOn(LED1);                    //LED1 turns on
wait(3);                            //for three seconds
turnLEDOff(LED1);                   //LED1 turns off

//FanMotor turns on
//Until Bumper is pressed
```

Program Design

- Continue programming while testing one behavior at a time.
  - Temporarily turn sections of code into comments using /* followed by */.

```
task main()
{
                                    //Program begins
/*
turnLEDOn(LED1);                    //LED1 turns on
wait(3);                            //for three seconds
turnLEDOff(LED1);                   //LED1 turns off
*/

startMotor(FanMotor, 127);          //FanMotor turns on
untilBump(Bumper);                  //Until Bumper is pressed
stopMotor(FanMotor);                //FanMotor turns off

//Light 2 (LED 2) turns on
```

Sample Programs

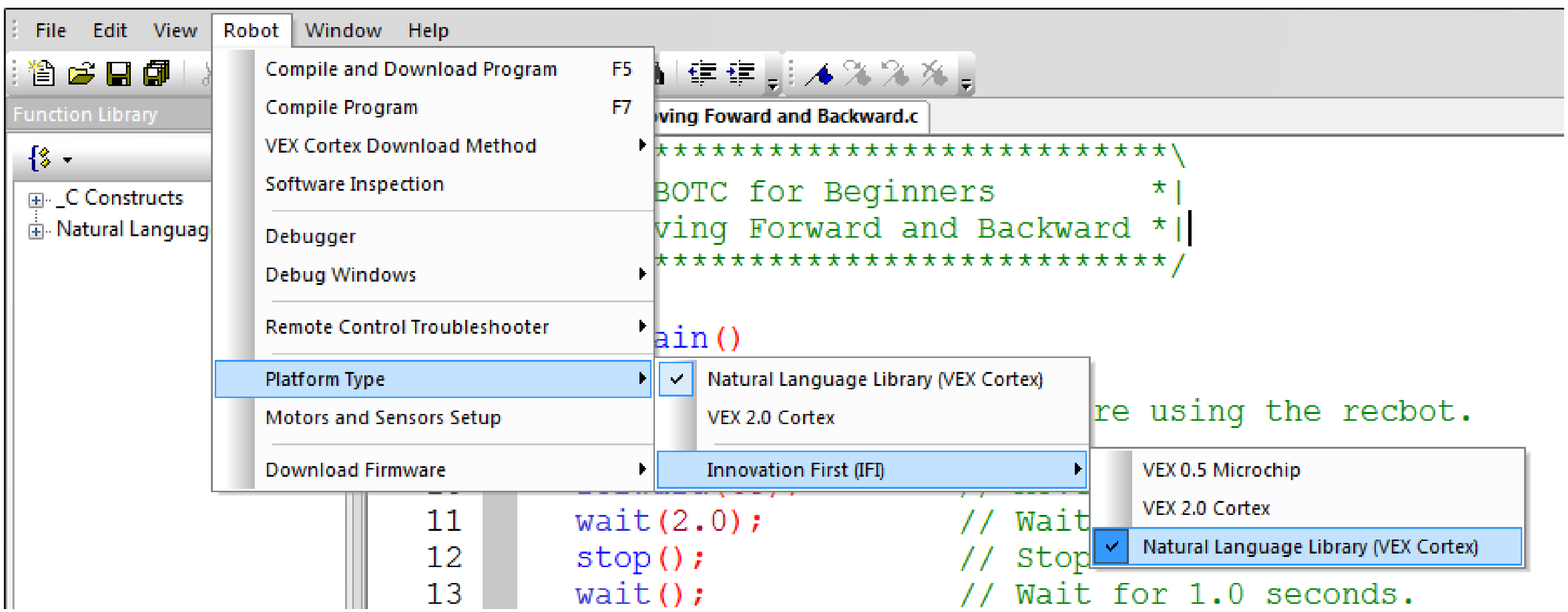


ROBOTC Natural Language

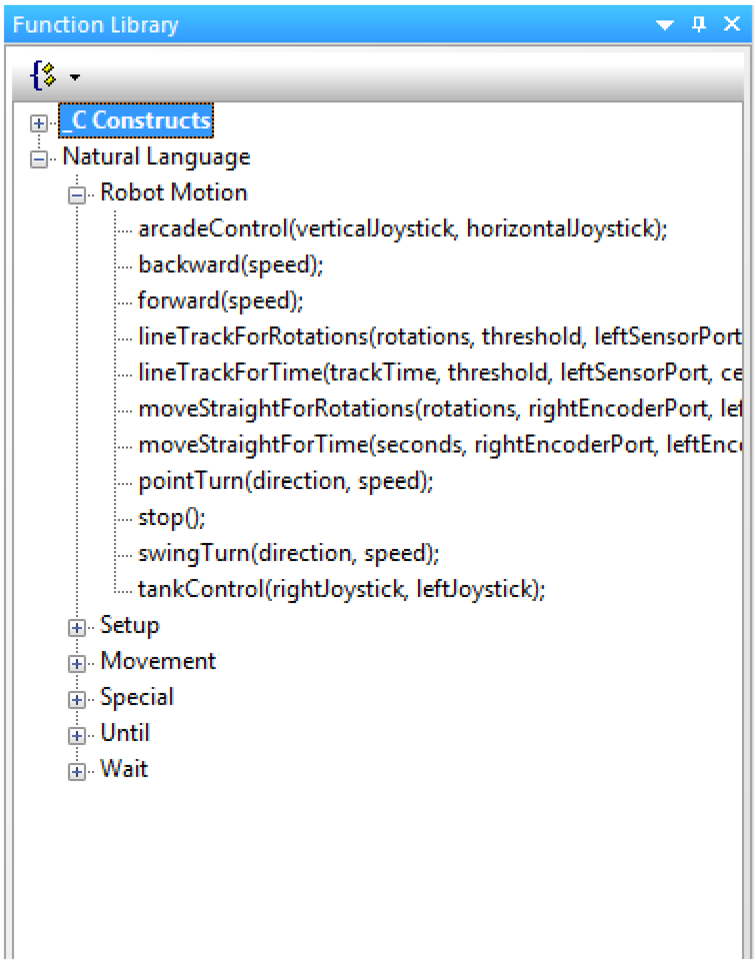
- New language developed specifically for PLTW
- Lines of code for common robot behaviors are consolidated into single commands
  - forward();
  - lineTrackforTime();
  - stop();
  - untilBump();

ROBOTC Natural Language

Natural Language is an additional Platform Type in ROBOTC.

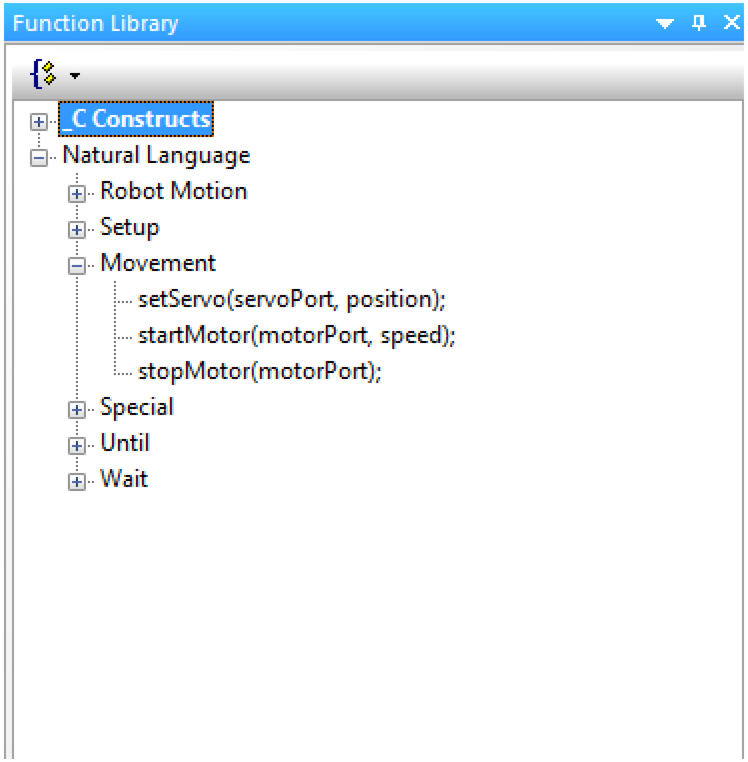


ROBOT Motion



Commands that cause the entire robot to perform a behavior

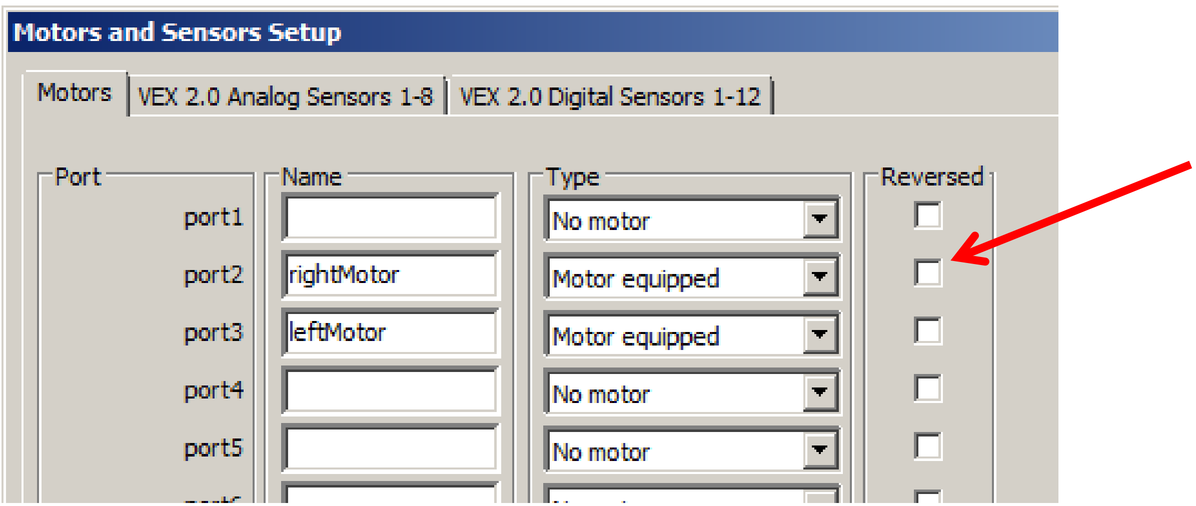
Movement



Commands that allow you to control individual motors / servos

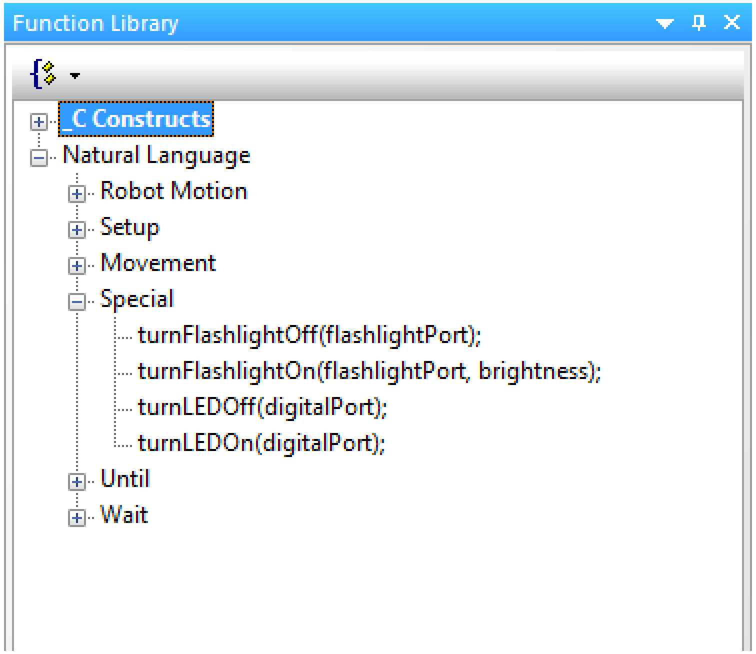
Motor Reversal

- Reversing motor polarity
  - Check Reverse in Motors and Setup
  - Change speed + / - in program



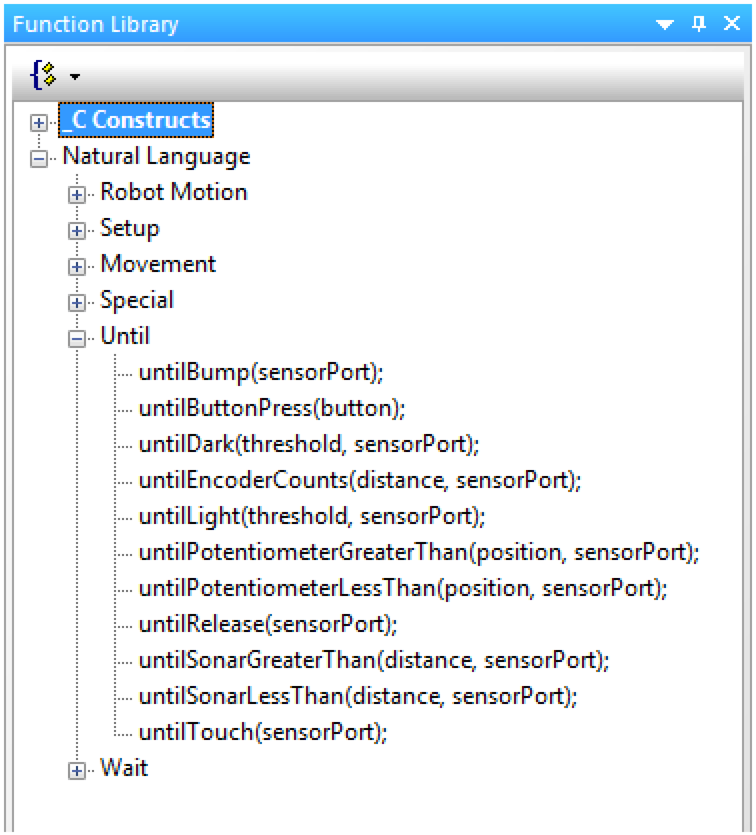

```
startMotor(rightMotor, 63);
startMotor(rightMotor, -63);
```

Special



Commands that control the more unique VEX Hardware – LEDs and Flashlights

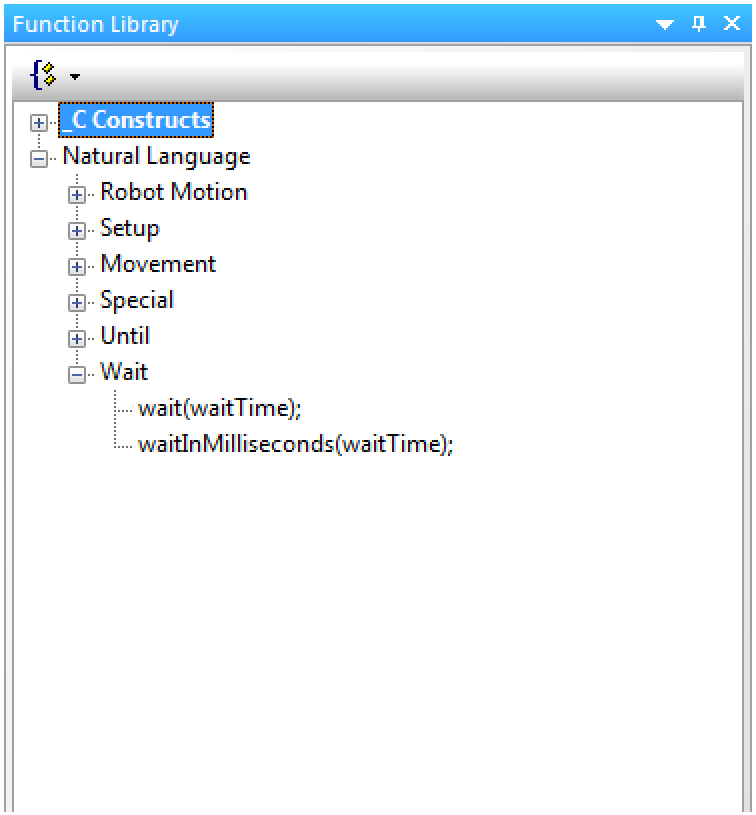
Until



Commands that allow you to create behaviors where the robot acts “until” a certain event

- Button Press
- Encoder Count Reached

Wait



Commands that wait for an elapsed amount of time in seconds or milliseconds

Wait States Accuracy

- Motor Speed is affected by battery power
  - If battery is fully charged, then motors move quickly.
  - If battery is low, then motors move slowly.
  - Device or robot will not move consistently as the battery power drains.

Touch Sensors

- Digital sensor – Pressed or Released

  1 = Pressed and 0 = Released

- Caution
  - Bouncing may occur when sensor is pressed or released
  - Value may bounce between 0 and 1 very briefly and quickly
  - Very brief wait can be inserted to reduce bouncing effect

```
task main()
{
  untilBump(bumper);
  wait(.05);
}
```

References

Carnegie Mellon Robotics Academy. (2011). ROBOTC. Retrieved from http://www.robotc.net