Reserved Words

Motors

Motor control and some fine-tuning commands.

`motor[output] = power;`
This turns the referenced VEX motor output either on or off and simultaneously sets its power level. The VEX has 8 motor outputs: `port1`, `port2`... up to `port8`. The VEX supports power levels from -127 (full reverse) to 127 (full forward). A power level of 0 will cause the motors to stop.

```
motor[port3]= 127;    //port3 - Full speed forward
motor[port2]= -127;   //port2 - Full speed reverse
```

`bMotorReflected[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 bMotorReflected[] 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. You can use "true" for `1` and "false" for `0`.

Before:

```
motor[port3]= 127;     //port3 - Full speed forward
motor[port2]= 127;     //port2 - Full speed reverse
```

After:

```
bMotorReflected[port2]= 1;  //Flip port2's direction
motor[port3]= 127;          //port3 - Full speed forward
motor[port2]= 127;          //motorA - Full speed forward
```

Timing

The VEX 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 value (where 1 = 1/1000th of a second). Maximum wait_time is 32768, or 32.768 seconds.

```
motor[port3]= 127;      //port3 - full speed forward
wait1Msec(2000);        //Wait 2 seconds
motor[port3]= 0;        //port3 - off
```

Reserved Words

`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 value (where 1 = 1/100th of a second). Maximum wait_time is 32768, or 327.68 seconds.

```
motor[port3]= 127;      //port3 - full speed forward
wait10Msec(200);        //Wait 2 seconds
motor[port3]= 0;        //port3 - 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 VEX 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 VEX 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.)
```

`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 VEX 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.)
```

Reserved Words

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

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

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

`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 VEX has 16 analog/digital inputs: `in1`, `in2...` to `in16`

```
if(SensorValue(in1) == 1) //If in1 (bumper) is pressed
{
  motor[port3] = 127;     //Motor Port 3 full speed forward
}
```

Type of Sensor	Digital/Analog?	Range of Values
Touch	Digital	0 or 1
Reflection (Ambient)	Analog	0 to 1023
Rotation (Older Encoder)	Digital	0 to 32676
Potentiometer	Analog	0 to 1023
Line Follower (Infrared)	Analog	0 to 1023
Sonar	Digital	-2, -1, and 1 to 253
Quadrature Encoder	Digital	-32678 to 32768
Digital In	Digital	0 or 1
Digital Out	Digital	0 or 1

Sounds

The VEX can play sounds and tones using an external piezoelectric speaker attached to a motor port.

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

```
PlayTone(220, 500); //Plays a 220hz tone for 1/2 second
```

Reserved Words

Radio Control

ROBOTC allows you to control your robot using input from the Radio Control Transmitter.

`bVexAutonomousMode`
Set the value to either `0` for radio enabled or `1` for radio disabled (autonomous mode). You can also use "true" for `1` and "false" for `0`.

```
bVexAutonomousMode = 0;   //enable radio control

bVexAutonomousMode = 1;    //disable radio control
```

`vexRT[joystick_channel]`
This command retrieves the value of the specified channel being transmitted.

If the RF receiver is plugged into Rx 1, the following values apply:

Control Port	Joystick Channel	Possible Values
Right Joystick, X-axis	Ch1	-127 to 127
Right Joystick, Y-axis	Ch2	-127 to 127
Left Joystick, Y-axis	Ch3	-127 to 127
Left Joystick, X-axis	Ch4	-127 to 127
Left Rear Buttons	Ch5	-127, 0, or 127
Right Rear Buttons	Ch6	-127, 0, or 127

If the RF receiver is plugged into Rx 2, the following values apply:

Control Port	Joystick Channel	Possible Values
Right Joystick, X-axis	Ch1Xmtr2	-127 to 127
Right Joystick, Y-axis	Ch2Xmtr2	-127 to 127
Left Joystick, Y-axis	Ch3Xmtr2	-127 to 127
Left Joystick, X-axis	Ch4Xmtr2	-127 to 127
Left Rear Buttons	Ch5Xmtr2	-127, 0, or 127
Right Rear Buttons	Ch6Xmtr2	-127, 0, or 127

```
bVexAutonomousMode = false;  //enable radio control
while(true)
{
  motor[port3] = vexRT[Ch3]; //right joystick, y-axis
                             //controls the motor on port 3
  motor[port2] = vexRT[Ch2]; //left joystick, y-axis
                             //controls the motor on port 2
}
```

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 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. An infinite while loop 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[port3]= 127;//...motor port3 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. The else condition is not required when using an if statement.

```
if(sensorValue(bumper) ==1)//the bumper is used as...
{                          //...the condition
 motor[port3]= 0;   //if it's pressed port3 stops
}
else
{
  motor[port3]= 127; //if it's not pressed port3 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
```

`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 ASCII character, specified between a set of single quotes.

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