1300 Henley Court
Pullman, WA 99163
509.334.6306
www.digilentinc.com

DMC60C Software Reference Manual

Revised 1/9/19
This manual applies to the DMC60C rev. E.1 with application firmware version 1.26 or newer and bootloader 1.9 or newer

Table of Contents

Table of Contents **1**

1. Initial Configuration (Web Configuration Utility) **3**

1.1 Devices Panel 3
1.1.1 Flashing Device LEDs 3
1.1.2 Advanced Settings 3
1.2 Configurable Parameters 4
1.2.1 Device 4
1.2.2 Current Limits 4
1.2.3 Limit Switches 4
1.2.4 PID Tuning 5
1.2.5 Soft Limits 6
1.2.6 Faults 6
1.3 Saving Parameters 6
1.4 Restore Defaults 7
1.5 Updating Firmware 7
1.6 Setting Brake/Coast 7

2. Creating a DMC60C Object **7**

2.1 FRC VS Code 2019 (C++/Java) 7
2.2 Labview 8

3. Configuring the DMC60C **8**

3.1 General Settings 8

3.2 Open Loop 9
3.3 Closed Loop 9
3.4 Programmatically Setting Web Parameters 9

4. Driving the DMC60C 10

4.1 Open Loop 10
4.2 Closed Loop 10
4.3 Follower Mode 10
4.4 Disable Motor 11

5. Getting the Status of the DMC60C 11

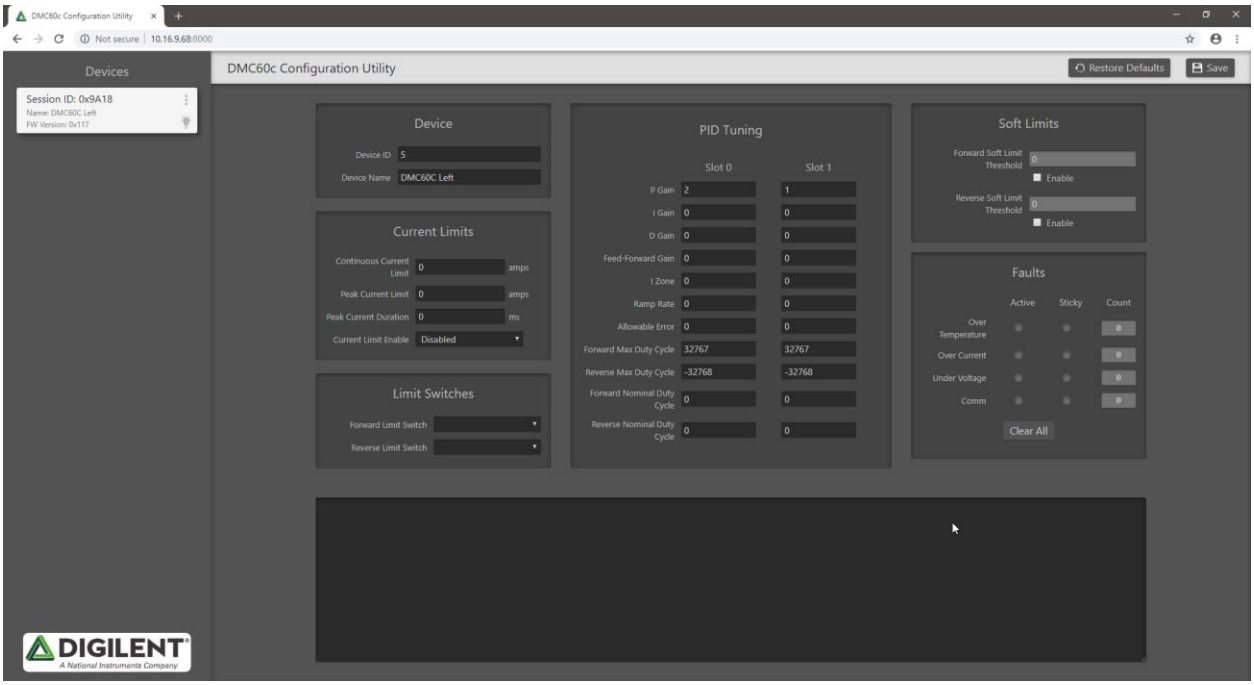
5.1 General Status 11
5.2 Limit Switches 11
5.3 Closed Loop Status 11
5.4 Fault Status 12

6. Tutorials 12

6.1 Setting correct motor and encoder direction 12
6.1.1 C++ (Robot.cpp) 13
6.1.2 Java (Robot.java) 14
6.1.3 Labview 14
6.2 Tuning closed loop parameters 15
6.2.1 Position Mode 15
6.2.2 Velocity Mode 15
6.3 Using current limiting 15
6.3.1 Fixed Current Limit 15
6.3.2 Variable Current Limit 15

1. Initial Configuration (Web Configuration Utility)

The DMC60C configuration utility is automatically installed on the roboRIO when installing the DMC60C API (https://github.com/Digilent/dmc60c-frc-api). This server is run each time the roboRIO boots up. By default it is hosted on port 8000. Users can access the web configuration server by opening an internet browser and typing "roborio-[teamnumber]-frc.local:8000", replacing [teamnumber] with the FRC team number. The roboRIO's IP address will also work and may respond faster. For example: 192.168.1.104:8000



The web configuration utility is used to change many of the DMC60C's parameters. These parameters are stored in nonvolatile memory and are preserved across power cycles. It is important to note that parameters are first saved in RAM and are not written to flash until no parameters have been updated for 15 seconds. This is necessary to reduce flash wear.

1.1 Devices Panel

The devices panel will show all DMC60C's connected to the roboRIO's CAN bus. Each device card will contain its Device ID, its name, and its firmware version.

1.1.1 Flashing Device LEDs

To identify which physical DMC60C you are configuring, click the light bulb icon on the device card. A command will be sent to the DMC60C to flash its LEDs in a rainbow pattern.

1.1.2 Advanced Settings

Clicking the three dot icon will reveal some advanced settings. It is not recommended to change these settings unless you know what you are doing. More information on these parameters can be found in the parameters section of the CAN protocol guide.

1.2 Configurable Parameters

1.2.1 Device

- **Device ID -** The device ID of the DMC60C on the CAN bus. This can be a number between 0 and 63. **No two DMC60Cs should share a device ID**. The DMC60C configuration utility will be able to tell them apart, but the robot application will not unless they have different device IDs.
- **Device Name -** The name of the DMC60C. This is used only in the web configuration utility to help keep track of which DMC60C is which.

1.2.2 Current Limits

- **Current Limit Enabled -** This will enable or disable current limiting.
- **Continuous Current Limit -** The maximum allowed continuous current limit. If load current exceeds the Peak Current Limit for longer than the Peak Current Duration and current limiting is enabled, then the load current will be limited to the value specified by the Continuous Current Limit. If the Continuous Current Limit is set to a value that's greater than or equal to the Peak Current Limit and current limiting is enabled, then the DMC60C will limit begin limiting the load current immediately after the first time that it detects that the Continuous Current Limit has been exceeded.
- **Peak Current Limit -** The maximum allowed peak current limit. If load current exceeds the Peak Current Limit for longer than the Peak Current Duration and current limiting is enabled, then the load current will be limited to the value specified by the Continuous Current Limit. If the Peak Current Duration is set to 0 and current limiting is enabled, then the DMC60C will begin applying the Continuous Current Limit immediately after the first time that it detects that the Peak Current Limit has been exceeded. If the Peak Current Limit is set to a value that's smaller than the Continuous Current Limit and current limiting is enabled, then the DMC60C will begin applying the Continuous Current Limit immediately after it detects that the Continuous Current Limit has been exceeded.
- **Peak Current Duration -** The duration that the motor will maintain the peak current. If load current exceeds the Peak Current Limit for longer than the Peak Current Duration and current limiting is enabled, then the load current will be limited to the value specified by the Continuous Current Limit. If the Peak Current Duration is set to 0 and current limiting is enabled, then the DMC60C will begin applying the Continuous Current Limit immediately after the first time that it detects that the Peak Current Limit has been exceeded. If the Peak Current Limit is set to a value that's smaller than the Continuous Current Limit and current limiting is enabled, then the DMC60C will begin applying the Continuous Current Limit immediately after it detects that the Continuous Current Limit has been exceeded.

1.2.3 Limit Switches

- **Forward Limit Switch -** This determines the mode of the forward limit switch. If enabled, the DMC60C will prevent the output from applying a positive voltage to the load when the limit switch is active (if it closes when normally open or opens when normally closed). The forward limit switch value is read from the FWDLIM pin on the DMC60C.
- **Reverse Limit Switch -** This determines the mode of the reverse limit switch. If enabled, the DMC60C will prevent the output from applying a negative voltage to the load when the limit switch is active (if it closes when normally open or opens when normally closed). The reverse limit switch value is read from the REVLIM pin on the DMC60C.

1.2.4 PID Tuning

There are two PID slots on the DMC60C. These can be configured separately and selected in software by calling SetPIDSlot(0 or 1).

- **P Gain -** The proportional gain constant. This constant is used during closed loop control to calculate a proportional increase or decrease in the throttle (duty cycle) due to the measured closed loop error. This value is used when driving the motor in one of the closed loop control modes (Velocity, Position, or Current).
- **I Gain -** The integral gain constant. This constant is used during closed loop control to calculate an integral increase or decrease in the throttle (duty cycle) due to the measured closed loop error. This value is used when driving the motor in one of the closed loop control modes (Velocity, Position, or Current).
- **D Gain -** The derivative gain constant. This constant is used during closed loop control to calculate a derivative increase or decrease in the throttle (duty cycle) due to the measured closed loop error. This value is used when driving the motor in one of the closed loop control modes (Velocity, Position, or Current).
- **Feed Forward Gain -** The feed forward gain constant. This constant is used during closed loop control to calculate the number of throttle units to contribute to the duty cycle as the proportion of the setpoint (target Velocity, Position, or Current) independent of the error. For example, if the target current is 20.0 amps and you want to apply 50% throttle for this setpoint then the feed forward gain would be set to $\frac{0.50 \times 32767}{20.0} = 819.175$. Convert this to fixed-point by multiplying by 65536. This results in a value of 0x03332CCC (hex), which is what should be sent to the DMC60C in the value field of the PARAMSET packet. The feed-forward term can be excluded from the PID calculations by specifying a value of 0 for the gain. This value is used when driving the motor in one of the closed loop control modes (Velocity, Position, or Current).
- **I Zone -** The integral accumulator limit. This is used to limit how large the integral accumulator can grow during closed loop control. The value sent to the DMC60C is converted to a 32-bit signed integer and used to set the positive and negative bounds of the integral accumulator. If the integral accumulator exceeds these bounds while PID calculations are performed, then the accumulator will be capped to IZone or -IZone. This provides a mechanism for combating integral windup. Setting a value of 0 will disable the limit and allow the integral accumulator to grow without bounds.
- **Ramp Rate -** The closed loop ramp rate. This specifies the maximum number of throttle units the output can change by each time the control loop executes in closed loop control mode (Velocity, Position, or Current). For example, If the closed loop ramp rate is set to 1000 and the PID update function determines that the throttle should be increased by 5000 units then the immediate throttle increase will be limited to 1000 units. If the next PID Update doesn't change the target throttle output value, the throttle will be increased by another 1000 units the next time the control loop executes. This process will continue until the target throttle is reached or a new throttle value is calculated. The control loop executes once every 500 µs. Therefore, specifying a closed loop ramp rate of 16 would result in it taking approximately 1.02 seconds to go from 0% throttle (0) to 100% throttle (32767). Specifying a value of 0 for the closed loop ramp rate disables throttling and allows the output to be immediately set to the target value.
- **Allowable Error -** The allowable closed loop error. This specifies the minimum error required for the PID controller to calculate a non-zero contribution to the output throttle based on the P, I, and D, terms. If the allowable error is set to a non-zero value and the measured error is less than the allowable error then the P, I, and D terms will contribute 0 throttle units to the output throttle calculation and the integral accumulator will be cleared. If the allowable error is set to 0 or the measured error exceeds the allowable error then P, I, and D terms are included in the output throttle calculation. The feed-forward gain constant, or F term, is included in the output throttle calculation regardless of the allowable error setting.

- **Forward Max Duty Cycle –** This parameter limits the maximum forward duty cycle that can be applied when this PID slot is in use. The value of this parameter should be between 0 and 32767. This value is used when driving the motor in one of the closed loop control modes (Velocity, Position, or Current).
- **Reverse Max Duty Cycle -** This parameter limits the maximum reverse duty cycle that can be applied when this PID slot is in use. The value of this parameter should be between -32768 and 0. This value is used when driving the motor in one of the closed loop control modes (Velocity, Position, or Current).
- **Forward Nominal Duty Cycle -** This parameter limits the smallest forward duty cycle that can be applied when the closed loop error exceeds the allowable closed loop error. The value of this parameter should be between 0 and 32767. This value is used when driving the motor in one of the closed loop control modes (Velocity, Position, or Current).
- **Reverse Nominal Duty Cycle –** This parameter limits the smallest reverse duty cycle that can be applied when the closed loop error exceeds the allowable closed loop error. The value of this parameter should be between -32768 and 0. This value is used when driving the motor in one of the closed loop control modes (Velocity, Position, or Current).

1.2.5 Soft Limits

- **Soft Forward Threshold -** The maximum position that the encoder can read in the forward direction. The units are native to the encoder that is connected to the DMC60C. Each time the DMC60C's control loop executes (500us) the current position of the encoder is read and compared to the soft forward limit threshold. If the current position is greater than the soft forward limit threshold and the soft forward limit is enabled, then the DMC60C's output will be prevented from applying a positive voltage to the load. Both positive and negative soft limit thresholds are valid.
- **Soft Reverse Threshold -** The maximum position that the encoder can read in the reverse direction. The units are native to the encoder that is connected to the DMC60C. Each time the DMC60C's control loop executes (500us) the current position of the encoder is read and compared to the soft forward limit threshold. If the current position is less than or equal to the soft forward limit threshold and the soft forward limit is enabled, then the DMC60C's output will be prevented from applying a negative voltage to the load. Both positive and negative soft limit thresholds are valid.

1.2.6 Faults

This section shows the current active and sticky faults that have occurred on the DMC60C. The sticky faults are saved in nonvolatile memory and are persistent across power cycles. The active and sticky lights represent whether there are active or sticky faults currently detected. The count box shows how many sticky faults have been logged since the faults were last cleared. Clicking the **Clear All** button will reset all sticky fault counts to 0.

- **Over Temperature** – Occurs when the temperature of the PCB reaches 100°C.
- **Under Voltage** – Occurs when the input voltage falls below 5.75 Volts (+/- 2%) for 5 or more seconds.
- **Communications** – Occurs when connection with the CAN bus is lost.

1.3 Saving Parameters

When you are ready to save your changes to the DMC60C parameters, click the 

button on the top panel. If the parameters were successfully updated, they will glow green for a few seconds. Otherwise they will turn red. You should see the changes logged in the console below. If any errors occurred, they will be shown in the console as well.

1.4 Restore Defaults

You can restore the parameters to their default values by clicking the Restore Defaults button. You can also restore the default values by holding down the BRAKE/COAST CAL LED button when powering on the DMC60C.

1.5 Updating Firmware

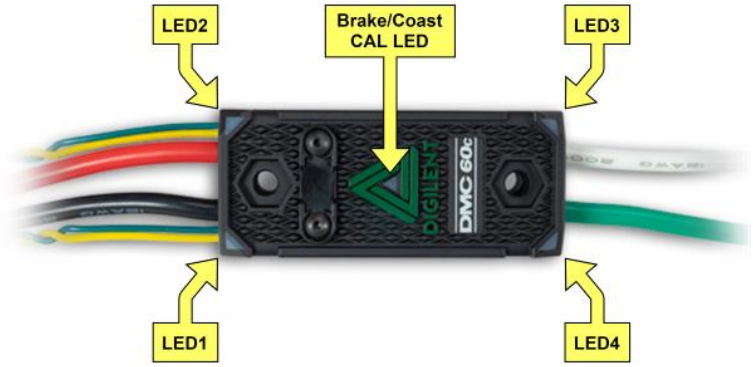
The DMC60C comes pre-loaded with both bootloader and application firmware images. The bootloader firmware version is fixed at version 1.9 (0x109) and is not meant to be updated. The application firmware should be 1.23 (0x117) out of the box. Updates found at the DMC60C product page on the reference website (https://reference.digilentinc.com/dmc-60c/start).

To update the firmware of a DMC60C:

1. Download the latest application firmware .hex file.
2. Right click the DMC60C's device card and click Update Firmware.
3. Select the downloaded firmware hex file.
4. Click update firmware.

If the update is successful, you will get an Update Successful message. If the firmware update is interrupted or unsuccessful, the DMC60C will stay in bootloader mode and the application firmware version will show 0xFFFF. At this point you should reattempt to update the firmware until it is successful.

1.6 Setting Brake/Coast



The Brake/Coast LED is located in the center of the triangle, which is located at the center of the housing, and is used to indicate the current Brake/Coast setting. When the center LED is off, the device is operating in coast mode. When the center LED is illuminated, the device is operating in brake mode. The Brake/Coast mode can be toggled by pressing down on the center of the triangle, and then releasing the button.

2. Creating a DMC60C Object

Creating a DMC60C object is similar in each of the supported languages (C++/Java/Labview). The API uses the DMC60C's device ID to specify which DMC60C to instantiate. This means each DMC60C must have different device IDs or there can be complications.

2.1 FRC VS Code 2019 (C++/Java)

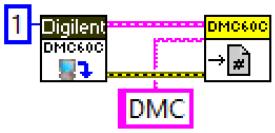
1. In FRC VS Code 2019, create a new project or open an existing VSCode FRC project.

2. Click the WPI logo on the top right.
3. Go to **Manage Vendor Libraries > Install new libraries (offline)**.
4. If the DMC60C API has been installed, you should see **Digilent-DMC60C**. Select that and press OK.
5. (C++)In your main robot.cpp, #include <digilent/dmc60/WPI_DMC60C.h>
6. (Java)In you main robot.java import com.dmc60.DMC60C
7. Create a new object:
    a. C++: DMC60::WPI_DMC60C * myDMC60C = new DMC60::WPI_DMC60C(deviceNum);
    b. Java: private DMC60C myDMC60C = new DMC60C(deviceNum);

2.2 Labview

1. In Labview, create a new robot project or open an existing 2019 robot project.
2. Open Target/Team Code/Begin.vi.
3. Press CTRL+E to open the block diagram view.
4. Right click to open the VI palette and go to **WPI Robotics Library > Third Party > DMC60C.**
    a. It can be useful to pin this subpalette for ease of use.
5. Click and drag the Open DMC60C VI to the block diagram.
6. Right click the device number input and create a constant. Change this to the DMC60C's device number.

7. In the DMC60C subpalette, open the WPI subpallete and add a DMC60C RefNum Registry Set.vi.
8. Connect the DMC60C output cluster to the input of the Registry Set vi.
9. Create a constant for the refnum name and give the DMC reference any name. (DMC in this picture)



10. To access this DMC60C device in another vi:
    a. Add a DMC60C RefNum Registry Get.vi
    b. Enter the refnum name as the input. The output will be the DMC60C reference cluster.

3. Configuring the DMC60C

The DMC60C has many configurable settings in the high-level API. These settings will typically be set when the robot code is initialized.

3.1 General Settings

- **SetInverted(bool isInverted)** – Inverts the motor direction.
    - **isInverted** – True to invert the motor, false otherwise.
- **invertEncoder(bool isInverted)** – Inverts the signals of the feedback sensor. This alleviates the need to swap the QEA and QEB signals when the quadrature encoder direction does not match the motor.
    - **isInverted** – True to invert the encoder, false otherwise.
- **configIndexActiveEdge(bool edge)** – Configures the index pin on the DMC60C to trigger on the rising or falling edge. Use this when using the index pin (pin 9) on the DMC60C**.**
    - **edge** – True to trigger on rising edge, false to trigger on falling edge.

3.2 Open Loop

- **configOpenLoopRampRate(unsigned short rampRate)** – Sets the ramp rate used in open loop modes. The ramp rate is the number of "throttle units" (0 being 0% duty cycle, 32767 being 100% duty cycle) that the output can change each time the control loop executes (500us). This function can be used to limit the rate at which the motor speeds up.
    - **rampRate** – How fast the motor can speed up. Smaller values result in slower acceleration and less current draw.

3.3 Closed Loop

- **configWheel(double wheelDiametermm, double gearRatio, int encoderTicks)** – Configures the parameters of the output of the motor. Calling this function is **required** when operating in closed loop velocity or position mode.
    - **wheelDiametermm** – The diameter of the wheel in millimeters. This is used in velocity and distance calculations.
    - **gearRatio** – The ratio of the encoder to the output. If the encoder is on the output of the gearbox, this value will be 1. If the encoder is on the input of a 12.75:1 gearbox, this value will be 12.75.
    - **encoderTicks** – The cycles per channel per revolution of the encoder in use.
- **setPIDSlot(unsigned int slot)** – Sets the active PID profile (slot).
    - **slot** – 0 to use PID slot 0, 1 to use PID slot 1.
- **configClosedLoopRampRate (unsigned int slot, unsigned int rampRate)** – Sets the closed loop ramp rate for the specified PID slot. This is separate from the open loop ramp rate but is functionally the same.
    - **slot** – The PID slot to configure.
    - **rampRate** – How fast the motor can speed up. Smaller values result in slower acceleration and less current draw.
- **configPositionReset(bool resetOnFwdLimit, bool resetOnRevLimit, bool resetOnIndex)** – Configures the DMC60C to reset (zero) its position count when one of the connected switches goes active. This will only be done on the active edge of the corresponding pin.

3.4 Programmatically Setting Web Parameters

Some of the parameters that are set in the web configuration utility can be programmatically set. These functions allow for dynamic configuration during operation.

- **configPID(unsigned int slot, float P, float I, float D, float F)** – Used to programmatically set PIDF values.
- **setContinuousCurrentLimit(double currentAmps)** – Sets the continuous current limit, described in section 2.2.2.
- **setPeakCurrentLimit(double currentAmps)** – Sets the peak current limit, described in section 2.2.2.
- **setPeakCurrentDuration(double currentAmps)** – Sets the peak current duration, described in section 2.2.2.
- **enableCurrentLimiting(bool enabled)** – Enables or disables current limiting.
    - **enabled** – True to enable, false to disable.
- **setLimitSwitches(bool overrideEnable, bool forwardSwitchEnable, bool reverseSwitchEnable)** – Overrides the limit switch settings to the specified configuration. This does not change the parameters, only overrides them.
    - **overrideEnable** – True to override the web configuration switch settings.
    - **forwardSwitchEnable** – If overrideEnable is true: enables or disables the forward Limit Switch.

    - **reverseSwitchEnable** – If overrideEnable is true: enables or disables the reverse Limit Switch.
- **setBrakeCoast(bool overrideEnable, bool brakeEnable)** – Overrides the hardware brake/coast configuration.
    - **overrideEnable** – If true, overrides the brake/coast setting set by the brake/coast LED on the DMC60C.
    - **brakeEnable** – If overrideEnable is true: True sets the DMC60C to brake when a neutral output is applied, false set it to coast when a neutral output is applied.

4. Driving the DMC60C

The DMC60C API takes a slightly different approach to driving the motors. In addition to the Set() function, there are separate drive***() functions. This was an attempt to make these functions and parameters more user friendly. The DMC60C is designed with a 104ms timeout. If no commands are sent to the DMC60C within 104ms, it will go into standby mode until a drive or disable command is sent. There are 6 drive modes available:

4.1 Open Loop

Open Loop drive modes do not require any additional hardware to use.

- **Set(double speed)** – Drives the output in percent voltage (duty cycle) mode. This function calls driveVoltage().
    - **speed** – The desired duty cycle, from -1 to 1.
- **driveVoltage(double percentVoltage)** – Drives the output in percent voltage (duty cycle) mode.
    - **percentVoltage** – The desired duty cycle, from -100 to 100 percent.
- **driveVoltageCompensation(double voltage)** – Drives the output with the desired voltage.
    - **voltage** – The signed voltage to send to the motor, from -inputvoltage to +inputvoltage. If the desired voltage exceeds the input voltage, the duty cycle will be limited to -100% or 100%.

4.2 Closed Loop

Closed Loop drive modes (excluding driveCurrent) require a quadrature encoder to work correctly. In addition to this, users must call the **configWheel()** function to properly set up the wheel parameters. Users should also call **setPIDSlot()** to select which PID profile to use.

- **driveCurrent(double currentAmps)** – Drives the output with the desired current.
    - **currentAmps** – The desired current in amperes to send to the motor.
- **zeroEncoderPosition()** – Sets the current position to 0.
- **drivePosition(double revolutions)** – Drives the output motor a certain number of revolutions relative to zero.
    - **revolutions** – Signed number of revolutions relative to zero to drive the motor.
- **driveDistance(double meters)** – Drives the output motor a certain number of meters relative to zero.
    - **meters** – Signed number of meters relative to zero to drive the motor.
- **driveVelocity(double metersPerSecond)** – Drives the output motor a certain speed in meters per second.
    - **metersPerSecond** – Signed speed in meters per second.

4.3 Follower Mode

In addition to directly driving the DMC60C, it is possible to set it into follower mode. A DMC60C in follower mode will attempt to match the duty cycle of another DMC60C on the CAN bus.

- **followerMode(int deviceNumbertoFollow)** – Sets the DMC60C to follow another DMC60C on the CAN bus.
    - **deviceNumbertoFollow** – The device number of the master DMC60C.

4.4 Disable Motor

A disable motor command is included which tells the DMC60C to apply a neutral output to the motor. This function should be called if no other drive commands are being sent to the DMC60C to feed the FRC watchdog and motor safety controller.

- **disableMotor() or Disable() or StopMotor()** – Sends a neutral output command to the DMC60C. If in brake mode, the DMC60C will hold the position of the motor in place. If in coast mode, the DMC60C will disable all current to the motor.

5. Getting the Status of the DMC60C

The status functions in the DMC60C API are all prefixed with “get” or “is”. This section will describe what each status message returns.

5.1 General Status

- **Get()** – Gets the current duty cycle scaled from -1 to 1.
- **getCurrentDutyCycle()** – Returns the current duty cycle applied to the output of the motor from -100% to 100%.
- **isEnabled()** – Returns whether or not the DMC60C is currently driving (not disabled).
- **GetInverted()** – Returns whether or not the motor is inverted.
- **isCurrentLimitActive()** – Returns whether or not the current limit is actively throttling the output current.
- **getBusVoltage()** - Returns the DMC60C's bus (battery) voltage in volts.
- **getAIN1Voltage()** - Returns the voltage applied to the AIN1 pin on the DMC60C in volts.
- **getLoadCurrent()** - Returns the load current of the DMC60C in amps. This is the current that is running through the motor.
- **getAmbientTemp()** - Returns the internal temperature of the DMC60C in °C.
- **getDeviceNumber()** – Returns the device number of the DMC60C.

5.2 Limit Switches

- **getFwdLimitSwitch()** – Returns the raw state of the forward limit switch.
- **getRevLimitSwitch()** – Returns the raw state of the reverse limit switch.
- **isFwdLimitSwitchActive()** – Returns true if the forward limit is active, false if not.
- **isRevLimitSwitchActive()** – Returns true if the reverse limit is active, false if not.
- **getFwdLimitSwitchStatus()** – Returns the complete status of the forward limit switch. (C++ only)
- **getRevLimitSwitchStatus()** – Returns the complete status of the reverse limit switch. (C++ only)

5.3 Closed Loop Status

- **getClosedLoopError()** – Returns the Closed Loop Error. Useful for PID tuning.
- **getRevolutionsTraveled()** – Returns the number of revolutions traveled relative to the zero position.
- **getDistanceTraveled()** – Returns the distance traveled in meters relative to the zero position.

- **getMetersPerSecond()** – Returns the current velocity in meters per second. This is measured in a 100ms window.
- **getEncoderPositionCount()** – Returns the raw encoder position count.
- **getEncoderVelocityCount()** – Returns the raw velocity count logged from the encoder in the past 100ms.
- **getP(unsigned int slot)** – Returns the P constant of the specified PID slot.
- **getI(unsigned int slot)** – Returns the I constant of the specified PID slot.
- **getD(unsigned int slot)** – Returns the D constant of the specified PID slot.
- **getF(unsigned int slot)** – Returns the F constant of the specified PID slot.

5.4 Fault Status

- **isOverTempFaultActive()** - Returns whether or not the over temperature fault is active.
- **isUnderVoltageFaultActive()** - Returns whether or not the under voltage fault is active.

6. Tutorials

This section will walk through a couple of common procedures to get the robot functioning properly.

6.1 Setting correct motor and encoder direction

In order to use limit switches and closed loop the encoder must be configured to work in phase with the motor In other words, the DMC60C needs to know that when the quadrature encoder count increases, the motor is moving forward. To do this, use the code below. Drive the motor forward manually using a joystick. If the motor is moving in the wrong direction, change the invert motor call in RobotInit() to **SetInverted (true)** and restart before continuing. Check the position count by reading the results from **getEncoderPositionCount()** on the Smart Dashboard (or the front panel in Labview). If the position count is negative, then the encoder needs to be inverted. To do this, change the invert encoder call to **invertEncoder(true)**. These settings are not stored in nonvolatile memory, so SetInverted() and invertEncoder() need to be called during robot initialization each time.

6.1.1 C++ (Robot.cpp)

```
/*----------------------------------------------------------------------------*/
/* Copyright (c) 2017-2018 FIRST. All Rights Reserved.                        */
/* Open Source Software - may be modified and shared by FRC teams. The code   */
/* must be accompanied by the FIRST BSD license file in the root directory of */
/* the project.                                                               */
/*----------------------------------------------------------------------------*/

#include "Robot.h"
#include <iostream>
#include <frc/smartdashboard/SmartDashboard.h>
#include <digilent/dmc60/WPI_DMC60C.h>
#include <frc/Joystick.h>
//DMC60C with device number 5.
DMC60::WPI_DMC60C * _dmc = new DMC60::WPI_DMC60C(5);
//Joystick on USB port 0.
frc::Joystick * _joy = new frc::Joystick(0);
void Robot::RobotInit() {
  //Change these depending on motor behavior.
  _dmc->SetInverted(false);
  _dmc->invertEncoder(false);
}
void Robot::TeleopInit() {
  _dmc->zeroEncoderPosition();
}
void Robot::TeleopPeriodic() {
  bool A = _joy->GetRawButton(1);
  double stick = -1*_joy->GetY();
//If A button is held
  if(A){
    //Drive -100 to 100 percent duty cycle.
    _dmc->drivePosition(stick*100);
    //Report position on Smart Dashboard.
    frc::SmartDashboard::PutNumber("Position", _dmc->getEncoderPositionCount());
  }
  else{
    _dmc->Disable();
  }
}
void Robot::TestPeriodic() {}
void Robot::RobotPeriodic() {}
void Robot::AutonomousInit() {}
void Robot::AutonomousPeriodic() {}
#ifndef RUNNING_FRC_TESTS
int main() { return frc::StartRobot<Robot>(); }
#endif
```

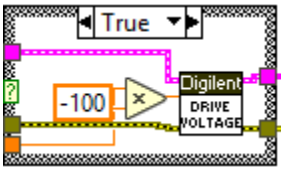
6.1.2 Java (Robot.java)

```
/*----------------------------------------------------------------------------*/
/* Copyright (c) 2017-2018 FIRST. All Rights Reserved.                        */
/* Open Source Software - may be modified and shared by FRC teams. The code   */
/* must be accompanied by the FIRST BSD license file in the root directory of */
/* the project.                                                               */
/*----------------------------------------------------------------------------*/

package frc.robot;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj.Joystick;
import com.dmc60c.DMC60C;
public class Robot extends TimedRobot {
  //DMC60C with device number 5
  private DMC60C _dmc = new DMC60C(5);
  //Joystick on USB port 0.
  private Joystick _joy = new Joystick(0);
  @Override
  public void robotInit() {
    //Invert motor and encoder based on the hardware configuration.
    _dmc.setInverted(false);
    _dmc.InvertEncoder(false);
  }
  @Override
  public void teleopInit() {
    //Reset the Encoder position to zero.
    _dmc.ZeroEncoderPosition();
  }
  @Override
  public void robotPeriodic() {}
  @Override
  public void teleopPeriodic() {
    boolean A = _joy.getRawButton(1);
    double stick = -1*_joy.getY();
  //If A button is held
    if(A){
      //Drive -100 to 100 percent duty cycle
      _dmc.DriveVoltage(stick*100);
      //Report position on Smart Dashboard.
      SmartDashboard.putNumber("Position", _dmc.GetEncoderPositionCount());
    }
    else{
      _dmc.disable();
    }
  }
}
```

6.1.3 Labview

Use the Position Mode Example project (found in dmc60c-frc-api\Examples\Labview\PositionModeExample). In teleOp.vi, replace the call to drivePosition with driveVoltage:



6.2 Tuning closed loop parameters

The DMC60C features two closed-loop control modes, position mode and velocity mode, that require a quadrature encoder. These modes require proper PID configuration to operate correctly. The PID constants can be set programmatically (section 4.4), although we recommend setting them in the web configuration utility (section 2.2.4). The PID constants will differ between position and velocity mode, so it is suggested that the settings be saved in separate PID slots.

6.2.1 Position Mode

To start, set the I, D, and F constants to 0 and the P constant to 0.1. Open the position mode example project (Found in dmc60c-frc-api\Examples). Edit the code to match your DMC60C's device number and encoder/wheel parameters. Add a call to getClosedLoopError in the TeleOp Periodic Function if it doesn't already exist:

In C++: frc::SmartDashboard::PutNumber("Closed Loop Error", _dmc->getClosedLoopError());

In Java: SmartDashboard.putNumber("Closed Loop Error", _dmc.GetClosedLoopError());

This will report the closed loop error to an FRC Smart Dashboard. It is recommended that these be mapped to graphs in the Smart Dashboard to make tuning easier. The Labview example project is already set up and does not require any additional changes.

Run the code and drive the motor forward or backwards in teleoperated mode. The motor should slowly move to the position relative to the joystick. Watch the closed loop error graph to see how the system reacts to your commands. The motor will likely come up short, so double the P constant until the motor responds quickly without too much overshoot. At this point the motor should be able to reach a set position. You can continue tuning the remaining closed loop parameters (I, D, F, ramp rate, max/nominal duty cycle, and allowable closed-loop error) to fine tune the acceleration and deceleration of the closed loop.

6.2.2 Velocity Mode

To start, set the feed-forward (F) gain to a small value. Open the velocity mode example project and set the proper DMC60C parameters in the code. Run the code and drive the motor forward. Watch the velocity closed loop error in the Smart Dashboard to see how the system reacts to your commands. Double the F gain until the system can reach the target velocity somewhat reliably. Next, tune the P gain so that the closed-loop can perform error correction. The remaining closed loop parameters can then be modified to fine tune the closed-loop response.

6.3 Using current limiting

Current limiting can be used to prevent brownouts when attempting to drive the motors connected to the DMC60C. Current limiting has 3 parameters to keep in mind: Continuous Current Limit, Peak Current Limit, and Peak Current Duration.

6.3.1 Fixed Current Limit

The continuous current limit parameter can be used alone to limit the current to a fixed value. This can be done by setting only the continuous current limit parameter while leaving the peak current limit and duration at 0.

6.3.2 Variable Current Limit

In some applications, a current spike is needed to get things moving initially. This can be accomplished by setting the peak current limit and duration to values other than 0. Once configured, the DMC60C will enable current

limiting when the output current exceeds the continuous current limit value. It will then allow current to flow up to the peak current limit for the duration specified by peak current duration. After the peak current duration time has passed, the current will be limited to the value specified by the continuous current limit. The DMC60C will stay in current limiting mode until the measured current is less than the continuous current limit AND the target duty cycle has decreased or is in the opposite direction.

7. Firmware Revisions

Version	Description
1.26	Added automatic current limit disable
1.23	Initial Release

8. Document Revisions

Revision	Description
1.0	Initial Release