Phoenix Framework Migration Guide

API Docs Java C++ | 2018 Documentation | Migration Guide | Create an issue | Check our Store

Migration Table

Description	HERO SDK v4/FRC-2017	Phoenix v5	Details
General			
class name change	**CANTalon**	**TalonSRX** (com.ctre.phoenix.motorcontrol.can.TalonSRX)	Talon SRX are predominantly used on CAN Bus, therefore the CAN prefix was redundant. For use in WPILib based applications, consider using the `WPI_TalonSRX` class.
class name change	**TalonSRX**	**PWMTalonSRX** (edu.wpi.first.wpilibj.PWMTalonSRX)	Talon SRX still supports PWM. This class is still maintained by the WPILIB team.
Set	set(value) / changeControlMode(mode)	set(mode,value)	Single param set() and changeControMode() have been replaced with a two parameter set(). Because value is dependent on the mode, having seperate functions caused confusion in previous seasons. For WPI classes that use the single param set, use the `WPI_TalonSRX` class.
timeoutMs parameter	talon.func(XXX)	talon.func(XXX, timeoutMs)	Some routines can optionally wait for a response from the device up to a timeout. If the response is never received, an error code is returned and the Drive Station will receive an error message. Pass 0 to avoid blocking at all. When initializing device on robot-boot, we recommend passing 10ms. When calling routines in the loop of the robot, pass zero to avoid blocking.
C++ Header	Individual files	"ctre/Phoenix.h"	Phoenix.h is a general header with several header includes and namespace scopes. This single header is sufficient for a majority of cases. If developer wants to avoid bringing in all of the namespaces, inspect the header for what includes are necessary and copy accordingly.

Brake Mode	enableBrakeMode	setNeutralMode	Funcion has been renamed.
Voltage Compensation			
...is not a Control Mode	setControlMode (CANTalon::kVoltage)	enableVoltage Compensation()	Voltage Compensation can be enabled/disabled in parallel to any control mode.
Setting compensation value for "full" output.		configVoltageComp Saturation (voltage, timeoutMs)	This is the voltage that the motor controller will compensate "full" output to. For example, if 10V is configured for the saturation point, when the motor controller is commanded 50%, it will modulate until the output is 5V. This applies to closed-loop modes as well.
Low level sensor API (without sensor selection)			
Accessing low level feedback sensor.	getPulseWidthPosition()	getSensorCollection(). getPulseWidthPosition()	Low level functions that access sensor data directly are now tucked under the sensor collection object.
Setting low level sensor.	talon.setEncPosition(newValue)	getSensorCollection(). setQuadraturePosition (pos, 10);	Same as above. Also note that "Enc" has changed to "Quadrature".
Direction / sensor phase / follower			
Direction/Sensor flipping	reverseSensor / reverseOutput / setInvert / SetSensorDirection / (similar reverse functions)	talon.SetInvert / talon.SetSensorPhase	Use sensor phase to align positive sensor velocity to positive motor output. Once this is done you will never have to call any reverse function for the purpose of alignment. Use SetInvert to choose what motor direction to apply when positive output is applied (green LEDs)
Sensor Units			
Position API	configEncoderCodesPerRev / configPotentiometerTurns	All Talon / Victor API uses sensor units	Positions are always in sensor units. Quadrature units are 4X measurements where X units represents X quadrature edges. Analog units are 1024 per voltage sweep from 0V to 3.3V (and wrapped back to 0V).
	getPosition	getSelectedSensorPosition	The return value is in sensor units. Sensor must be selected using configSelectedFeedbackSensor()/ Multiply by (1/SensorUnitsPerRotation) to convert into rotations.
	configEncoderCodesPerRev /	All Talon / Victor API uses	Velocity units are always change in sensor units per 100ms, regardless of configuration.

Velocity API	configPotentiometerTurns	sensor units per 100ms	Tachometer is always in units per 100ms, where each units is 1/1024 of a rotation.
	getSpeed	getSelectedSensorVelocity	The return value is in units per 100ms for all sensor types. Sensor must be selected using configSelectedFeedbackSensor()/ Multiply by (600/SensorUnitsPerRotation) to convert into RPM.
Peak and Nominal outputs			
	configNominalOutputVoltage / configPeakOutputVoltage /	configPeakOutputForward / configPeakOutputReverse / configNominalOutputForward / configNominalOutputReverse	The inputs are now [-1,+1] which represents [-100%,+100], and not based on voltage. This was never the case as last year the inputs where naively divided by 12.
Factory Default - Press and hold B/C button on boot.			
	Resets PWM Calibration	Resets PWM Calibration / Resets persistent CAN settings, EXCEPT for Device ID and Neutral Brake.	This should be done on any Talon when replacing/swapping/troubleshooting. Because of the large number of persistent config settings, it is simpler to default all peak/nominal/measurement/etc configs first and only deliberately set the desired parameters in code.
Selecting sensor	SetFeedbackDevice	ConfigSelectedFeedbackSensor	All Config* routines are persistent