SPARK MAX - C++ Documentation

Generated by Doxygen 1.8.15

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 rev::CANAnalog Class Reference

Inherits rev::CANSensor.

Public Types

- enum AnalogMode { **kAbsolute** = 0, **kRelative** = 1 }

Public Member Functions

- CANAnalog (CANSparkMax &device, AnalogMode mode)
- **CANAnalog** (CANAnalog &&)=default
- CANAnalog & **operator=** (CANAnalog &&)=default
- double GetVoltage ()
- double GetPosition ()
- double GetVelocity ()
- CANError SetPositionConversionFactor (double factor)
- double GetPositionConversionFactor ()
- CANError SetVelocityConversionFactor (double factor)
- double GetVelocityConversionFactor ()
- CANError SetAverageDepth (uint32_t depth)
- CANError SetMeasurementPeriod (uint32_t period_ms)
- uint32_t GetAverageDepth ()
- uint32_t GetMeasurementPeriod ()
- CANError SetInverted (bool inverted) override
- bool GetInverted () const override

Protected Member Functions

- int GetID () const override

Additional Inherited Members

3.1.1 Member Enumeration Documentation

3.1.1.1 AnalogMode

```
enum rev::CANAnalog::AnalogMode  [strong]
```

Analog sensors have the ability to either be absolute or relative. By default, GetAnalog() will return an absolute analog sensor, but it can also be configured to be a relative sensor instead.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CANAnalog()

```
CANAnalog::CANAnalog (
            CANSparkMax & device,
            AnalogMode mode )  [explicit]
```

Constucts a CANAnalog.

Parameters

device	The Spark Max to which the analog sensor is attached.

3.1.3 Member Function Documentation

3.1.3.1 GetAverageDepth()

```
uint32_t CANAnalog::GetAverageDepth ( )
```

Get the averafe sampling depth for a quadrature encoder.

Returns

The average sampling depth

3.1.3.2 GetID()

```
int CANAnalog::GetID ( ) const  [override], [protected], [virtual]
```

Get the ID of the sensor that is connected to the SparkMax through the encoder port on the front of the controller (not the top port).

Returns

The ID of the sensor

Implements rev::CANSensor.

3.1.3.3 GetInverted()

```
bool CANAnalog::GetInverted ( ) const  [override], [virtual]
```

Get the phase of the CANSensor. This will just return false if the user tries to get inverted while the SparkMax is Brushless and using the hall effect sensor.

Returns

The phase of the encoder

Implements rev::CANSensor.

3.1.3.4 GetMeasurementPeriod()

```
uint32_t CANAnalog::GetMeasurementPeriod ( )
```

Get the number of samples for reading from a quadrature encoder. This value sets the number of samples in the average for velocity readings.

Returns

Measurement period in microseconds

3.1.3.5 GetPosition()

```
double CANAnalog::GetPosition ( )
```

Get the position of the motor. Returns value in the native unit of 'volt' by default, and can be changed by a scale factor using setPositionConversionFactor().

Returns

Position of the sensor in volts

3.1.3.6 GetPositionConversionFactor()

```
double CANAnalog::GetPositionConversionFactor ( )
```

Get the current conversion factor for the position of the analog sensor.

Returns

Analog position conversion factor

3.1.3.7 GetVelocity()

```
double CANAnalog::GetVelocity ( )
```

Get the velocity of the motor. Returns value in the native units of 'volts per second' by default, and can be changed by a scale factor using setVelocityConversionFactor().

Returns

Velocity of the sensor in volts per second

3.1.3.8 GetVelocityConversionFactor()

```
double CANAnalog::GetVelocityConversionFactor ( )
```

Get the current conversion factor for the velocity of the analog sensor.

Returns

Analog velocity conversion factor

3.1.3.9 GetVoltage()

```
double CANAnalog::GetVoltage ( )
```

Get the voltage of the analog sensor.

Returns

Voltage of the sensor

3.1.3.10 SetAverageDepth()

```
CANError CANAnalog::SetAverageDepth (
            uint32_t depth )
```

Set the average sampling depth for a quadrature encoder. This value sets the number of samples in the average for velocity readings. This can be any value from 1 to 64.

When the SparkMax controller is in Brushless mode, this will not change any behavior.

Parameters

depth	The average sampling depth between 1 and 64 (default)

Returns

CANError.kOK if successful

3.1.3.11 SetInverted()

```
CANError CANAnalog::SetInverted (
            bool inverted )  [override], [virtual]
```

Set the phase of the CANSensor so that it is set to be in phase with the motor itself. This only works for quadrature encoders. This will throw an error if the user tries to set inverted while the SparkMax is Brushless and using the hall effect sensor.

Parameters

inverted	The phase of the encoder

Returns

CANError.kOK if successful

Implements rev::CANSensor.

3.1.3.12 SetMeasurementPeriod()

```
CANError CANAnalog::SetMeasurementPeriod (
            uint32_t period_ms )
```

Set the measurement period for velocity measurements of a quadrature encoder. When the SparkMax controller is in Brushless mode, this will not change any behavior.

The basic formula to calculate velocity is change in positon / change in time. This parameter sets the change in time for measurement.

Parameters

period_us	Measurement period in milliseconds. This number may be between 1 and 100 (default).

Returns

CANError.kOK if successful

3.1.3.13 SetPositionConversionFactor()

```
CANError CANAnalog::SetPositionConversionFactor (
            double factor )
```

Set the conversion factor for the position of the analog sensor. By default, revolutions per volt is 1. Changing the position conversion factor will also change the position units.

Parameters

factor	The conversion factor which will be multiplied by volts

Returns

CANError Set to CANError.kOK if successful

3.1.3.14 SetVelocityConversionFactor()

```
CANError CANAnalog::SetVelocityConversionFactor (
            double factor )
```

Set the conversion factor for the veolocity of the analog sensor. By default, revolutions per volt second is 1. Changing the velocity conversion factor will also change the velocity units.

Parameters

factor	The conversion factor which will be multipled by volts per second

Returns

CANError Set to CANError.kOK is successful

The documentation for this class was generated from the following files:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANAnalog.h
- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/cpp/CANAnalog.cpp

3.2 rev::CANDigitalInput Class Reference

Public Types

- enum **LimitSwitch** { **kForward**, **kReverse** }
- enum **LimitSwitchPolarity** { **kNormallyOpen** = 0, **kNormallyClosed** = 1 }

Public Member Functions

- CANDigitalInput (CANSparkMax &device, LimitSwitch limitSwitch, LimitSwitchPolarity polarity)
- **CANDigitalInput** (CANDigitalInput &&)=default
- CANDigitalInput & **operator=** (CANDigitalInput &&)=default
- bool Get () const
- CANError EnableLimitSwitch (bool enable)
- bool IsLimitSwitchEnabled ()

3.2.1 Constructor & Destructor Documentation

3.2.1.1 CANDigitalInput()

```
CANDigitalInput::CANDigitalInput (
            CANSparkMax & device,
            LimitSwitch limitSwitch,
            LimitSwitchPolarity polarity )  [explicit]
```

Constructs a CANDigitalInput.

Parameters

device	The Spark Max to which the limit switch is attached.
limitSwitch	Whether this is forward or reverse limit switch.
polarity	Whether the limit switch is normally open or normally closed.

3.2.2 Member Function Documentation

3.2.2.1 EnableLimitSwitch()

```
CANError CANDigitalInput::EnableLimitSwitch (
            bool enable )
```

Enables or disables controller shutdown based on limit switch.

3.2.2.2 Get()

```
bool CANDigitalInput::Get ( ) const
```

Get the value from a digital input channel.

Retrieve the value of a single digital input channel from a motor controller. This method will return the state of the limit input based on the selected polarity, whether or not it is enabled.

3.2.2.3 IsLimitSwitchEnabled()

```
bool CANDigitalInput::IsLimitSwitchEnabled ( )
```

Returns true if limit switch is enabled.

The documentation for this class was generated from the following files:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANDigitalInput.h
- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/cpp/CANDigitalInput.cpp

3.3 rev::CANEncoder Class Reference

Inherits rev::CANSensor.

Public Member Functions

- CANEncoder (CANSparkMax &device, SensorType sensorType, int cpr)
- **CANEncoder** (CANEncoder &&rhs)
- CANEncoder & **operator=** (CANEncoder &&rhs)
- double GetPosition ()
- double GetVelocity ()
- CANError SetPosition (double position)
- CANError SetPositionConversionFactor (double factor)
- CANError SetVelocityConversionFactor (double factor)
- double GetPositionConversionFactor ()
- double GetVelocityConversionFactor ()
- CANError SetAverageDepth (uint32_t depth)
- CANError SetMeasurementPeriod (uint32_t period_ms)
- uint32_t GetAverageDepth ()
- uint32_t GetMeasurementPeriod ()
- uint32_t GetCPR ()
- CANError SetInverted (bool inverted) override
- bool GetInverted () const override
- CANError GetLastError ()

Protected Member Functions

- int GetID () const override

Additional Inherited Members

3.3.1 Constructor & Destructor Documentation

3.3.1.1 CANEncoder()

```
CANEncoder::CANEncoder (
            CANSparkMax & device,
            SensorType sensorType,
            int cpr )  [explicit]
```

Constructs a CANEncoder.

Parameters

device	The Spark Max to which the encoder is attached.
sensorType	The encoder type for the motor: kHallEffect or kQuadrature
cpr	The counts per revolution of the encoder

3.3.2 Member Function Documentation

3.3.2.1 GetAverageDepth()

```
uint32_t CANEncoder::GetAverageDepth ( )
```

Get the averafe sampling depth for a quadrature encoder.

Returns

The average sampling depth

3.3.2.2 GetCPR()

```
uint32_t CANEncoder::GetCPR ( )
```

Get the counts per revolution of the quadrature encoder.

Returns

Counts per revolution

3.3.2.3 GetID()

```
int CANEncoder::GetID ( ) const  [override], [protected], [virtual]
```

Get the ID of the sensor that is connected to the SparkMax through the encoder port on the front of the controller (not the top port).

Returns

The ID of the sensor

Implements rev::CANSensor.

3.3.2.4 GetInverted()

```
bool CANEncoder::GetInverted ( ) const  [override], [virtual]
```

Get the phase of the CANSensor. This will just return false if the user tries to get inverted while the SparkMax is Brushless and using the hall effect sensor.

Returns

The phase of the encoder

Implements rev::CANSensor.

3.3.2.5 GetLastError()

```
CANError rev::CANEncoder::GetLastError ( )
```

Returns the last error generated

3.3.2.6 GetMeasurementPeriod()

```
uint32_t CANEncoder::GetMeasurementPeriod ( )
```

Get the number of samples for reading from a quadrature encoder. This value sets the number of samples in the average for velocity readings.

Returns

Measurement period in microseconds

3.3.2.7 GetPosition()

```
double CANEncoder::GetPosition ( )
```

Get the position of the motor. This returns the native units of 'rotations' by default, and can be changed by a scale factor using setPositionConversionFactor().

Returns

Number of rotations of the motor

3.3.2.8 GetPositionConversionFactor()

```
double CANEncoder::GetPositionConversionFactor ( )
```

Get the conversion factor for position of the encoder. Multiplied by the native output units to give you position

Returns

The conversion factor for position

3.3.2.9 GetVelocity()

```
double CANEncoder::GetVelocity ( )
```

Get the velocity of the motor. This returns the native units of 'RPM' by default, and can be changed by a scale factor using setVelocityConversionFactor().

Returns

Number the RPM of the motor

3.3.2.10 GetVelocityConversionFactor()

```
double CANEncoder::GetVelocityConversionFactor ( )
```

Get the conversion factor for velocity of the encoder. Multiplied by the native output units to give you velocity

Returns

The conversion factor for velocity

3.3.2.11 SetAverageDepth()

```
CANError CANEncoder::SetAverageDepth (
            uint32_t depth )
```

Set the average sampling depth for a quadrature encoder. This value sets the number of samples in the average for velocity readings. This can be any value from 1 to 64.

When the SparkMax controller is in Brushless mode, this will not change any behavior.

Parameters

depth	The average sampling depth between 1 and 64 (default)

Returns

CANError.kOK if successful

3.3.2.12 SetInverted()

```
CANError CANEncoder::SetInverted (
            bool inverted )  [override], [virtual]
```

Set the phase of the CANSensor so that it is set to be in phase with the motor itself. This only works for quadrature encoders. This will throw an error if the user tries to set inverted while the SparkMax is Brushless and using the hall effect sensor.

Parameters

inverted	The phase of the encoder

Returns

CANError.kOK if successful

Implements rev::CANSensor.

3.3.2.13 SetMeasurementPeriod()

```
CANError CANEncoder::SetMeasurementPeriod (
            uint32_t period_ms )
```

Set the measurement period for velocity measurements of a quadrature encoder. When the SparkMax controller is in Brushless mode, this will not change any behavior.

The basic formula to calculate velocity is change in positon / change in time. This parameter sets the change in time for measurement.

Parameters

period_us	Measurement period in milliseconds. This number may be between 1 and 100 (default).

Returns

CANError.kOK if successful

3.3.2.14 SetPosition()

```
CANError CANEncoder::SetPosition (
            double position )
```

Set the position of the encoder.

Parameters

position	Number of rotations of the motor

Returns

CANError Set to CANError.kOK if successful

3.3.2.15 SetPositionConversionFactor()

```
CANError CANEncoder::SetPositionConversionFactor (
            double factor )
```

Set the conversion factor for position of the encoder. Multiplied by the native output units to give you position

Parameters

factor	The conversion factor to multiply the native units by

Returns

CANError Set to CANError.kOK if successful

3.3.2.16 SetVelocityConversionFactor()

```
CANError CANEncoder::SetVelocityConversionFactor (
            double factor )
```

Set the conversion factor for velocity of the encoder. Multiplied by the native output units to give you velocity

Parameters

factor	The conversion factor to multiply the native units by

Returns

CANError Set to CANError.kOK if successful

The documentation for this class was generated from the following files:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANEncoder.h
- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/cpp/CANEncoder.cpp

3.4 rev::CANPIDController Class Reference

Public Types

- enum **AccelStrategy** { **kTrapezoidal** = 0, **kSCurve** = 1 }
- enum **ArbFFUnits** { **kVoltage** = 0, **kPercentOut** = 1 }

Public Member Functions

- CANPIDController (CANSparkMax &device)
- **CANPIDController** (CANPIDController &&)=default
- CANPIDController & **operator=** (CANPIDController &&)=default
- CANError SetReference (double value, ControlType ctrl, int pidSlot=0, double arbFeedforward=0, ArbFFUnits arbFFUnits=ArbFFUnits::kVoltage)
- CANError SetP (double gain, int slotID=0)
- CANError SetI (double gain, int slotID=0)
- CANError SetD (double gain, int slotID=0)
- CANError SetDFilter (double gain, int slotID=0)
- CANError SetFF (double gain, int slotID=0)
- CANError SetIZone (double IZone, int slotID=0)
- CANError SetOutputRange (double min, double max, int slotID=0)
- double GetP (int slotID=0)
- double GetI (int slotID=0)
- double GetD (int slotID=0)
- double GetDFilter (int slotID=0)
- double GetFF (int slotID=0)
- double GetIZone (int slotID=0)
- double GetOutputMin (int slotID=0)
- double GetOutputMax (int slotID=0)
- CANError SetSmartMotionMaxVelocity (double maxVel, int slotID=0)
- CANError SetSmartMotionMaxAccel (double maxAccel, int slotID=0)
- CANError SetSmartMotionMinOutputVelocity (double minVel, int slotID=0)
- CANError SetSmartMotionAllowedClosedLoopError (double allowedErr, int slotID=0)
- CANError SetSmartMotionAccelStrategy (AccelStrategy accelStrategy, int slotID=0)
- double GetSmartMotionMaxVelocity (int slotID=0)
- double GetSmartMotionMaxAccel (int slotID=0)
- double GetSmartMotionMinOutputVelocity (int slotID=0)
- double GetSmartMotionAllowedClosedLoopError (int slotID=0)
- AccelStrategy GetSmartMotionAccelStrategy (int slotID=0)
- CANError SetIMaxAccum (double iMaxAccum, int slotID=0)
- double GetIMaxAccum (int slotID=0)
- CANError SetIAccum (double iAccum)
- double GetIAccum ()
- CANError SetFeedbackDevice (const CANSensor &sensor)

3.4.1 Constructor & Destructor Documentation

3.4.1.1 CANPIDController()

```
CANPIDController::CANPIDController (
            CANSparkMax & device )  [explicit]
```

Constructs a CANPIDController.

Parameters

device	The Spark Max this object configures.

3.4.2 Member Function Documentation

3.4.2.1 GetD()

```
double CANPIDController::GetD (
            int slotID = 0 )
```

Get the Derivative Gain constant of the PIDF controller on the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

double D Gain value

3.4.2.2 GetDFilter()

```
double CANPIDController::GetDFilter (
            int slotID = 0 )
```

Get the Derivative Filter constant of the PIDF controller on the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

double D Filter value

3.4.2.3 GetFF()

```
double CANPIDController::GetFF (
            int slotID = 0 )
```

Get the Feed-forward Gain constant of the PIDF controller on the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

double F Gain value

3.4.2.4 GetI()

```
double CANPIDController::GetI (
            int slotID = 0 )
```

Get the Integral Gain constant of the PIDF controller on the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

double I Gain value

3.4.2.5 GetIAccum()

```
double CANPIDController::GetIAccum ( )
```

Get the I accumulator of the PID controller. This is useful when wishing to see what the I accumulator value is to help with PID tuning

Returns

The value of the I accumulator

3.4.2.6 GetIMaxAccum()

```
double CANPIDController::GetIMaxAccum (
            int slotID = 0 )
```

Get the maximum I accumulator of the PID controller. This value is used to constrain the I accumulator to help manage integral wind-up

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

The max value to contrain the I accumulator to

3.4.2.7 GetIZone()

```
double CANPIDController::GetIZone (
            int slotID = 0 )
```

Get the IZone constant of the PIDF controller on the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

double IZone value

3.4.2.8 GetOutputMax()

```
double CANPIDController::GetOutputMax (
            int slotID = 0 )
```

Get the max output of the PIDF controller on the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

double max value

3.4.2.9 GetOutputMin()

```
double CANPIDController::GetOutputMin (
            int slotID = 0 )
```

Get the min output of the PIDF controller on the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

double min value

3.4.2.10 GetP()

```
double CANPIDController::GetP (
            int slotID = 0 )
```

Get the Proportional Gain constant of the PIDF controller on the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

double P Gain value

3.4.2.11 GetSmartMotionAccelStrategy()

```
CANPIDController::AccelStrategy CANPIDController::GetSmartMotionAccelStrategy (
            int slotID = 0 )
```

Get the acceleration strategy used to control acceleration on the motor. The current strategy is trapezoidal motion profiling.

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

The acceleration strategy to use for the automatically generated motion profile

3.4.2.12 GetSmartMotionAllowedClosedLoopError()

```
double CANPIDController::GetSmartMotionAllowedClosedLoopError (
            int slotID = 0 )
```

Get the allowed closed loop error of SmartMotion mode. This value is how much deviation from your setpoint is tolerated and is useful in preventing oscillation around your setpoint.

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

The allowed deviation for your setpoint vs actual position in rotations

3.4.2.13 GetSmartMotionMaxAccel()

```
double CANPIDController::GetSmartMotionMaxAccel (
            int slotID = 0 )
```

Get the maximum acceleration of the SmartMotion mode. This is the accleration that the motor velocity will increase at until the max velocity is reached

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

The maxmimum acceleration for the motion profile in RPM per second

3.4.2.14 GetSmartMotionMaxVelocity()

```
double CANPIDController::GetSmartMotionMaxVelocity (
            int slotID = 0 )
```

Get the maximum velocity of the SmartMotion mode. This is the velocity that is reached in the middle of the profile and is what the motor should spend most of its time at

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

The maxmimum cruise velocity for the motion profile in RPM

3.4.2.15 GetSmartMotionMinOutputVelocity()

```
double CANPIDController::GetSmartMotionMinOutputVelocity (
            int slotID = 0 )
```

Get the mimimum velocity of the SmartMotion mode. Any requested velocities below this value will be set to 0.

Parameters

slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

The minimum velocity for the motion profile in RPM

3.4.2.16 SetD()

```
CANError CANPIDController::SetD (
            double gain,
            int slotID = 0 )
```

Set the Derivative Gain constant of the PIDF controller on the SPARK MAX. This uses the Set Parameter API and should be used infrequently. The parameter does not presist unless burnFlash() is called. The recommended method to configure this parameter is use to SPARK MAX GUI to tune and save parameters.

Parameters

gain	The derivative gain value, must be positive
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to REV_OK if successful

3.4.2.17 SetDFilter()

```
CANError CANPIDController::SetDFilter (
            double gain,
            int slotID = 0 )
```

Set the Derivative Filter constant of the PIDF controller on the SPARK MAX. This uses the Set Parameter API and should be used infrequently. The parameter does not presist unless burnFlash() is called.

Parameters

gain	The derivative filter value, must be a positive number between 0 and 1
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to REV_OK if successful

3.4.2.18 SetFeedbackDevice()

```
CANError CANPIDController::SetFeedbackDevice (
            const CANSensor & sensor )
```

Set the controller's feedback device.

The default feedback device is assumed to be the integrated encoder. This is used to changed to another feedback device for the controller, such as an analog sensor.

If there is a limited range on the feedback sensor that should be observed by the PIDController, it can be set by calling SetFeedbackSensorRange() on the sensor object.

Parameters

sensor	The sensor to be used as a feedback device

Returns

CANError set to kOK if successful

3.4.2.19 SetFF()

```
CANError CANPIDController::SetFF (
            double gain,
            int slotID = 0 )
```

Set the Feed-froward Gain constant of the PIDF controller on the SPARK MAX. This uses the Set Parameter API and should be used infrequently. The parameter does not presist unless burnFlash() is called. The recommended method to configure this parameter is use to SPARK MAX GUI to tune and save parameters.

Parameters

gain	The feed-forward gain value
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to REV_OK if successful

3.4.2.20 SetI()

```
CANError CANPIDController::SetI (
            double gain,
            int slotID = 0 )
```

Set the Integral Gain constant of the PIDF controller on the SPARK MAX. This uses the Set Parameter API and should be used infrequently. The parameter does not presist unless burnFlash() is called. The recommended method to configure this parameter is use to SPARK MAX GUI to tune and save parameters.

Parameters

gain	The integral gain value, must be positive
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to REV_OK if successful

3.4.2.21 SetIAccum()

```
CANError CANPIDController::SetIAccum (
            double iAccum )
```

Set the I accumulator of the PID controller. This is useful when wishing to force a reset on the I accumulator of the PID controller. You can also preset values to see how it will respond to certain I characteristics

To use this function, the controller must be in a closed loop control mode by calling setReference()

Parameters

iAccum	The value to set the I accumulator to

Returns

CANError Set to kOK if successful

3.4.2.22 SetIMaxAccum()

```
CANError CANPIDController::SetIMaxAccum (
            double iMaxAccum,
            int slotID = 0 )
```

Configure the maximum I accumulator of the PID controller. This value is used to constrain the I accumulator to help manage integral wind-up

Parameters

iMaxAccum	The max value to contrain the I accumulator to
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to kOK if successful

3.4.2.23 SetIZone()

```
CANError CANPIDController::SetIZone (
            double IZone,
            int slotID = 0 )
```

Set the IZone range of the PIDF controller on the SPARK MAX. This value specifies the range the |error| must be within for the integral constant to take effect.

This uses the Set Parameter API and should be used infrequently. The parameter does not presist unless burn↩ Flash() is called. The recommended method to configure this parameter is to use the SPARK MAX GUI to tune and save parameters.

Parameters

gain	The IZone value, must be positive. Set to 0 to disable
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to REV_OK if successful

3.4.2.24 SetOutputRange()

```
CANError CANPIDController::SetOutputRange (
            double min,
            double max,
            int slotID = 0 )
```

Set the min amd max output for the closed loop mode.

This uses the Set Parameter API and should be used infrequently. The parameter does not presist unless burn↩ Flash() is called. The recommended method to configure this parameter is to use the SPARK MAX GUI to tune and save parameters.

Parameters

min	Reverse power minimum to allow the controller to output
max	Forward power maximum to allow the controller to output
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to REV_OK if successful

3.4.2.25 SetP()

```
CANError CANPIDController::SetP (
            double gain,
            int slotID = 0 )
```

Set the Proportional Gain constant of the PIDF controller on the SPARK MAX. This uses the Set Parameter API and should be used infrequently. The parameter does not presist unless burnFlash() is called. The recommended method to configure this parameter is use to SPARK MAX GUI to tune and save parameters.

Parameters

gain	The proportional gain value, must be positive
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to REV_OK if successful

3.4.2.26 SetReference()

```
CANError CANPIDController::SetReference (
            double value,
```

```
            ControlType ctrl,
            int pidSlot = 0,
            double arbFeedforward = 0,
            ArbFFUnits arbFFUnits = ArbFFUnits::kVoltage )
```

Set the controller reference value based on the selected control mode.

Parameters

value	The value to set depending on the control mode. For basic duty cycle control this should be a value between -1 and 1 Otherwise: Voltage Control: Voltage (volts) Velocity Control: Velocity (RPM) Position Control: Position (Rotations) Current Control: Current (Amps). The units can be changed for position and velocity by a scale factor using setPositionConversionFactor().
ctrl	Is the control type
pidSlot	for this command
arbFeedforward	A value from -32.0 to 32.0 which is a voltage applied to the motor after the result of the specified control mode. The units for the parameter is Volts. This value is set after the control mode, but before any current limits or ramp rates.

Returns

CANError Set to REV_OK if successful

3.4.2.27 SetSmartMotionAccelStrategy()

```
CANError CANPIDController::SetSmartMotionAccelStrategy (
            CANPIDController::AccelStrategy accelStrategy,
            int slotID = 0 )
```

Coming soon. Configure the acceleration strategy used to control acceleration on the motor. The current strategy is trapezoidal motion profiling.

Parameters

accelStrategy	The acceleration strategy to use for the automatically generated motion profile
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to kOK if successful

3.4.2.28 SetSmartMotionAllowedClosedLoopError()

```
CANError CANPIDController::SetSmartMotionAllowedClosedLoopError (
            double allowedErr,
            int slotID = 0 )
```

Configure the allowed closed loop error of SmartMotion mode. This value is how much deviation from your setpoint is tolerated and is useful in preventing oscillation around your setpoint.

Parameters

allowedErr	The allowed deviation for your setpoint vs actual position in rotations
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to kOK if successful

3.4.2.29 SetSmartMotionMaxAccel()

```
CANError CANPIDController::SetSmartMotionMaxAccel (
            double maxAccel,
            int slotID = 0 )
```

Configure the maximum acceleration of the SmartMotion mode. This is the accleration that the motor velocity will increase at until the max velocity is reached

Parameters

maxAccel	The maxmimum acceleration for the motion profile in RPM per second
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to kOK if successful

3.4.2.30 SetSmartMotionMaxVelocity()

```
CANError CANPIDController::SetSmartMotionMaxVelocity (
            double maxVel,
            int slotID = 0 )
```

Configure the maximum velocity of the SmartMotion mode. This is the velocity that is reached in the middle of the profile and is what the motor should spend most of its time at

Parameters

maxVel	The maxmimum cruise velocity for the motion profile in RPM
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to kOK if successful

3.4.2.31 SetSmartMotionMinOutputVelocity()

```
CANError CANPIDController::SetSmartMotionMinOutputVelocity (
            double minVel,
            int slotID = 0 )
```

Configure the mimimum velocity of the SmartMotion mode. Any requested velocities below this value will be set to 0.

Parameters

minVel	The minimum velocity for the motion profile in RPM
slotID	Is the gain schedule slot, the value is a number between 0 and 3. Each slot has its own set of gain values and can be changed in each control frame using SetReference().

Returns

CANError Set to kOK if successful

The documentation for this class was generated from the following files:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANPIDController.h
- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/cpp/CANPIDController.cpp

3.5 rev::CANSensor Class Reference

Inherited by rev::CANAnalog, and rev::CANEncoder.

Public Member Functions

- **CANSensor** (CANSparkMax &device)
- virtual CANError SetInverted (bool inverted)=0
- virtual bool GetInverted () const =0

Protected Member Functions

- virtual int GetID () const =0

Protected Attributes

- CANSparkMax * m_device

Friends

- class **CANPIDController**

3.5.1 Member Function Documentation

3.5.1.1 GetID()

```
virtual int rev::CANSensor::GetID ( ) const  [protected], [pure virtual]
```

Get the ID of the sensor that is connected to the SparkMax through the encoder port on the front of the controller (not the top port).

Returns

The ID of the sensor

Implemented in rev::CANEncoder, and rev::CANAnalog.

3.5.1.2 GetInverted()

```
virtual bool rev::CANSensor::GetInverted ( ) const  [pure virtual]
```

Get the phase of the CANSensor. This will just return false if the user tries to get the inversion of the hall effect.

Implemented in rev::CANEncoder, and rev::CANAnalog.

3.5.1.3 SetInverted()

```
virtual CANError rev::CANSensor::SetInverted (
            bool inverted )  [pure virtual]
```

Set the phase of the CANSensor so that it is set to be in phase with the motor itself. This only works for quadrature encoders and analog sensors. This will throw an error if the user tries to set the inversion of the hall effect.

Implemented in rev::CANEncoder, and rev::CANAnalog.

3.5.2 Member Data Documentation

3.5.2.1 m_device

```
CANSparkMax* rev::CANSensor::m_device  [protected]
```

The range of the feedback sensor can be specified if the sensor has a limited range, such as an absolute encoder or analog device. The default range is -infinity to infinity. This is NOT the same as setting soft limits, as this will only check to ensure that setpoint commands for the controller fall within this range.

Parameters

minRange	The lower bound on the range of the sensor
maxRange	The upper bound on the range of the sensor

The documentation for this class was generated from the following file:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANSensor.h

3.6 rev::CANSparkMax Class Reference

Inherits rev::CANSparkMaxLowLevel.

Classes

- struct ExternalFollower

Public Types

- enum **IdleMode** { **kCoast** = 0, **kBrake** = 1 }
- enum **InputMode** { **kPWM** = 0, **kCAN** = 1 }
- enum **SoftLimitDirection** { **kForward**, **kReverse** }
- enum **FaultID** {
  kBrownout = 0, **kOvercurrent** = 1, **kIWDTReset** = 2, **kMotorFault** = 3,
  kSensorFault = 4, **kStall** = 5, **kEEPROMCRC** = 6, **kCANTX** = 7,
  kCANRX = 8, **kHasReset** = 9, **kDRVFault** = 10, **kOtherFault** = 11,
  kSoftLimitFwd = 12, **kSoftLimitRev** = 13, **kHardLimitFwd** = 14, **kHardLimitRev** = 15 }

Public Member Functions

- CANSparkMax (int deviceID, MotorType type)
- ~CANSparkMax () override=default
- void Set (double speed) override
- double Get () const override
- void SetInverted (bool isInverted) override
- bool GetInverted () const override
- void Disable () override
- void StopMotor () override
- void **PIDWrite** (double output) override
- CANEncoder GetEncoder (SensorType sensorType=SensorType::kHallSensor, int cpr=0)
- CANAnalog GetAnalog (CANAnalog::AnalogMode mode=CANAnalog::AnalogMode::kAbsolute)
- CANPIDController GetPIDController ()
- CANDigitalInput GetForwardLimitSwitch (CANDigitalInput::LimitSwitchPolarity polarity)
- CANDigitalInput GetReverseLimitSwitch (CANDigitalInput::LimitSwitchPolarity polarity)
- CANError SetSmartCurrentLimit (unsigned int limit)
- CANError SetSmartCurrentLimit (unsigned int stallLimit, unsigned int freeLimit, unsigned int limitRPM=20000)
- CANError SetSecondaryCurrentLimit (double limit, int limitCycles=0)
- CANError SetIdleMode (IdleMode mode)

- IdleMode GetIdleMode ()
- CANError EnableVoltageCompensation (double nominalVoltage)
- CANError DisableVoltageCompensation ()
- double GetVoltageCompensationNominalVoltage ()
- CANError SetOpenLoopRampRate (double rate)
- CANError SetClosedLoopRampRate (double rate)
- double GetOpenLoopRampRate ()
- double GetClosedLoopRampRate ()
- CANError Follow (const CANSparkMax &leader, bool invert=false)
- CANError Follow (ExternalFollower leader, int deviceID, bool invert=false)
- bool IsFollower ()
- uint16_t GetFaults ()
- uint16_t GetStickyFaults ()
- bool GetFault (FaultID faultID)
- bool GetStickyFault (FaultID faultID)
- double GetBusVoltage ()
- double GetAppliedOutput ()
- double GetOutputCurrent ()
- double GetMotorTemperature ()
- CANError ClearFaults ()
- CANError BurnFlash ()
- CANError SetCANTimeout (int milliseconds)
- CANError EnableSoftLimit (SoftLimitDirection direction, bool enable)
- bool IsSoftLimitEnabled (SoftLimitDirection direction)
- CANError SetSoftLimit (SoftLimitDirection direction, double limit)
- double GetSoftLimit (SoftLimitDirection direction)
- CANError GetLastError ()

Static Public Attributes

- static constexpr ExternalFollower **kFollowerDisabled** {0, 0}
- static constexpr ExternalFollower **kFollowerSparkMax** {0x2051800, 26}
- static constexpr ExternalFollower **kFollowerPhoenix** {0x2040080, 27}

Additional Inherited Members

3.6.1 Constructor & Destructor Documentation

3.6.1.1 CANSparkMax()

```
CANSparkMax::CANSparkMax (
            int deviceID,
            MotorType type )  [explicit]
```

Create a new SPARK MAX Controller

Parameters

deviceID	The device ID.
type	The motor type connected to the controller. Brushless motors must be connected to their matching color and the hall sensor plugged in. Brushed motors must be connected to the Red and Black terminals only.

3.6.1.2 ~CANSparkMax()

```
rev::CANSparkMax::~CANSparkMax ( )  [override], [default]
```

Closes the SPARK MAX Controller

3.6.2 Member Function Documentation

3.6.2.1 BurnFlash()

```
CANError CANSparkMax::BurnFlash ( )
```

Writes all settings to flash.

3.6.2.2 ClearFaults()

```
CANError CANSparkMax::ClearFaults ( )
```

Clears all non-sticky faults.

Sticky faults must be cleared by resetting the motor controller.

3.6.2.3 Disable()

```
void CANSparkMax::Disable ( )  [override]
```

Common interface for disabling a motor.

3.6.2.4 DisableVoltageCompensation()

```
CANError CANSparkMax::DisableVoltageCompensation ( )
```

Disables the voltage compensation setting for all modes on the SPARK MAX.

Returns

CANError Set to CANError.kOK if successful

3.6.2.5 EnableSoftLimit()

```
CANError CANSparkMax::EnableSoftLimit (
            SoftLimitDirection direction,
            bool enable )
```

Enable soft limits

Parameters

direction	the direction of motion to restrict
enable	set true to enable soft limits

3.6.2.6 EnableVoltageCompensation()

```
CANError CANSparkMax::EnableVoltageCompensation (
            double nominalVoltage )
```

Sets the voltage compensation setting for all modes on the SPARK MAX and enables voltage compensation.

Parameters

nominalVoltage	Nominal voltage to compensate output to

Returns

CANError Set to CANError.kOK if successful

3.6.2.7 Follow() [1/2]

```
CANError CANSparkMax::Follow (
            const CANSparkMax & leader,
            bool invert = false )
```

Causes this controller's output to mirror the provided leader.

Only voltage output is mirrored. Settings changed on the leader do not affect the follower.

Following anything other than a CAN SPARK MAX is not officially supported.

Parameters

leader	The motor controller to follow.
invert	Set the follower to output opposite of the leader

3.6.2.8 Follow() [2/2]

```
CANError CANSparkMax::Follow (
            ExternalFollower leader,
```

```
            int deviceID,
            bool invert = false )
```

Causes this controller's output to mirror the provided leader.

Only voltage output is mirrored. Settings changed on the leader do not affect the follower.

Following anything other than a CAN SPARK MAX is not officially supported.

Parameters

leader	The type of motor controller to follow (Talon SRX, Spark Max, etc.).
deviceID	The CAN ID of the device to follow.
invert	Set the follower to output opposite of the leader

3.6.2.9 Get()

```
double CANSparkMax::Get ( ) const  [override]
```

Common interface for getting the current set speed of a speed controller.

Returns

The current set speed. Value is between -1.0 and 1.0.

3.6.2.10 GetAnalog()

```
CANAnalog CANSparkMax::GetAnalog (
            CANAnalog::AnalogMode mode = CANAnalog::AnalogMode::kAbsolute )
```

Returns an object for interfacing with a connected analog sensor. By default, the AnalogMode is set to kAbsolute, thus treating the sensor as an absolute sensor.

3.6.2.11 GetAppliedOutput()

```
double CANSparkMax::GetAppliedOutput ( )
```

Returns motor controller's output duty cycle.

3.6.2.12 GetBusVoltage()

```
double CANSparkMax::GetBusVoltage ( )
```

Returns the voltage fed into the motor controller.

3.6.2.13 GetClosedLoopRampRate()

```
double CANSparkMax::GetClosedLoopRampRate ( )
```

Get the configured closed loop ramp rate

This is the maximum rate at which the motor controller's output is allowed to change.

Returns

rampte rate time in seconds to go from 0 to full throttle.

3.6.2.14 GetEncoder()

```
CANEncoder CANSparkMax::GetEncoder (
            SensorType sensorType = SensorType::kHallSensor,
            int cpr = 0 )
```

Returns an object for interfacing with the integrated encoder.

The default encoder type is assumed to be the hall effect for brushless. This can be modified for brushed DC to use an quadrature encoder.

3.6.2.15 GetFault()

```
bool CANSparkMax::GetFault (
            FaultID faultID )
```

Returns whether the fault with the given ID occurred.

3.6.2.16 GetFaults()

```
uint16_t CANSparkMax::GetFaults ( )
```

Returns fault bits.

3.6.2.17 GetForwardLimitSwitch()

```
CANDigitalInput CANSparkMax::GetForwardLimitSwitch (
            CANDigitalInput::LimitSwitchPolarity polarity )
```

Returns an object for interfacing with the integrated forward limit switch.

Parameters

polarity	Whether the limit switch is normally open or normally closed.

3.6.2.18 GetIdleMode()

```
CANSparkMax::IdleMode CANSparkMax::GetIdleMode ( )
```

Gets the idle mode setting for the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Returns

IdleMode Idle mode setting

3.6.2.19 GetInverted()

```
bool CANSparkMax::GetInverted ( ) const  [override]
```

Common interface for returning the inversion state of a speed controller.

This call has no effect if the controller is a follower.

Returns

isInverted The state of inversion, true is inverted.

3.6.2.20 GetLastError()

```
CANError CANSparkMax::GetLastError ( )
```

All device errors are tracked on a per thread basis for all devices in that thread. This is meant to be called immediately following another call that has the possibility of throwing an error to validate if an error has occurred.

Returns

the last error that was generated.

3.6.2.21 GetMotorTemperature()

```
double CANSparkMax::GetMotorTemperature ( )
```

Returns the motor temperature in Celsius.

3.6.2.22 GetOpenLoopRampRate()

```
double CANSparkMax::GetOpenLoopRampRate ( )
```

Get the configured open loop ramp rate

This is the maximum rate at which the motor controller's output is allowed to change.

Returns

rampte rate time in seconds to go from 0 to full throttle.

3.6.2.23 GetOutputCurrent()

```
double CANSparkMax::GetOutputCurrent ( )
```

Returns motor controller's output current in Amps.

3.6.2.24 GetPIDController()

```
CANPIDController CANSparkMax::GetPIDController ( )
```

Returns an object for interfacing with the integrated PID controller.

3.6.2.25 GetReverseLimitSwitch()

```
CANDigitalInput CANSparkMax::GetReverseLimitSwitch (
            CANDigitalInput::LimitSwitchPolarity polarity )
```

Returns an object for interfacing with the integrated reverse limit switch.

Parameters

polarity	Whether the limit switch is normally open or normally closed.

3.6.2.26 GetSoftLimit()

```
double CANSparkMax::GetSoftLimit (
            CANSparkMax::SoftLimitDirection direction )
```

Get the soft limit setting in the controller

Parameters

direction	the direction of motion to restrict

Returns

position soft limit setting of the controller

3.6.2.27 GetStickyFault()

```
bool CANSparkMax::GetStickyFault (
            FaultID faultID )
```

Returns whether the sticky fault with the given ID occurred.

3.6.2.28 GetStickyFaults()

```
uint16_t CANSparkMax::GetStickyFaults ( )
```

Returns sticky fault bits.

3.6.2.29 GetVoltageCompensationNominalVoltage()

```
double CANSparkMax::GetVoltageCompensationNominalVoltage ( )
```

Get the configured voltage compensation nominal voltage value

Returns

The nominal voltage for voltage compensation mode.

3.6.2.30 IsFollower()

```
bool CANSparkMax::IsFollower ( )
```

Returns whether the controller is following another controller

Returns

True if this device is following another controller false otherwise

3.6.2.31 IsSoftLimitEnabled()

```
bool CANSparkMax::IsSoftLimitEnabled (
            CANSparkMax::SoftLimitDirection direction )
```

Returns true if the soft limit is enabled.

3.6.2.32 Set()

```
void CANSparkMax::Set (
            double speed )  [override]
```

Common interface for setting the speed of a speed controller.

Parameters

speed	The speed to set. Value should be between -1.0 and 1.0.

3.6.2.33 SetCANTimeout()

```
CANError CANSparkMax::SetCANTimeout (
            int milliseconds )
```

Sets timeout for sending CAN messages. A timeout of 0 also means that error handling will be done automatically by registering calls and waiting for responses, rather than needing to call GetLastError().

Parameters

milliseconds	The timeout in milliseconds.

3.6.2.34 SetClosedLoopRampRate()

```
CANError CANSparkMax::SetClosedLoopRampRate (
            double rate )
```

Sets the ramp rate for closed loop control modes.

This is the maximum rate at which the motor controller's output is allowed to change.

Parameters

rate	Time in seconds to go from 0 to full throttle.

3.6.2.35 SetIdleMode()

```
CANError CANSparkMax::SetIdleMode (
            IdleMode mode )
```

Sets the idle mode setting for the SPARK MAX.

Parameters

mode	Idle mode (coast or brake).

3.6.2.36 SetInverted()

```
void CANSparkMax::SetInverted (
            bool isInverted )  [override]
```

Common interface for inverting direction of a speed controller.

This call has no effect if the controller is a follower.

Parameters

isInverted	The state of inversion, true is inverted.

3.6.2.37 SetOpenLoopRampRate()

```
CANError CANSparkMax::SetOpenLoopRampRate (
            double rate )
```

Sets the ramp rate for open loop control modes.

This is the maximum rate at which the motor controller's output is allowed to change.

Parameters

rate	Time in seconds to go from 0 to full throttle.

3.6.2.38 SetSecondaryCurrentLimit()

```
CANError CANSparkMax::SetSecondaryCurrentLimit (
            double limit,
            int limitCycles = 0 )
```

Sets the secondary current limit in Amps.

The motor controller will disable the output of the controller briefly if the current limit is exceeded to reduce the current. This limit is a simplified 'on/off' controller. This limit is enabled by default but is set higher than the default Smart Current Limit.

The time the controller is off after the current limit is reached is determined by the parameter limitCycles, which is the number of PWM cycles (20kHz). The recommended value is the default of 0 which is the minimum time and is part of a PWM cycle from when the over current is detected. This allows the controller to regulate the current close to the limit value.

The total time is set by the equation
```
 t = (50us - t0) + 50us * limitCycles
t = total off time after over current
t0 = time from the start of the PWM cycle until over current is detected
```

Parameters

limit	The current limit in Amps.
limitCycles	The number of additional PWM cycles to turn the driver off after overcurrent is detected.

3.6.2.39 SetSmartCurrentLimit() [1/2]

```
CANError CANSparkMax::SetSmartCurrentLimit (
            unsigned int limit )
```

Sets the current limit in Amps.

The motor controller will reduce the controller voltage output to avoid surpassing this limit. This limit is enabled by default and used for brushless only. This limit is highly recommended when using the NEO brushless motor.

The NEO Brushless Motor has a low internal resistance, which can mean large current spikes that could be enough to cause damage to the motor and controller. This current limit provides a smarter strategy to deal with high current draws and keep the motor and controller operating in a safe region.

Parameters

limit	The current limit in Amps.

3.6.2.40 SetSmartCurrentLimit() [2/2]

```
CANError CANSparkMax::SetSmartCurrentLimit (
            unsigned int stallLimit,
            unsigned int freeLimit,
            unsigned int limitRPM = 20000 )
```

Sets the current limit in Amps.

The motor controller will reduce the controller voltage output to avoid surpassing this limit. This limit is enabled by default and used for brushless only. This limit is highly recommended when using the NEO brushless motor.

The NEO Brushless Motor has a low internal resistance, which can mean large current spikes that could be enough to cause damage to the motor and controller. This current limit provides a smarter strategy to deal with high current draws and keep the motor and controller operating in a safe region.

The controller can also limit the current based on the RPM of the motor in a linear fashion to help with controllability in closed loop control. For a response that is linear the entire RPM range leave limit RPM at 0.

Parameters

stallLimit	The current limit in Amps at 0 RPM.
freeLimit	The current limit at free speed (5700RPM for NEO).
limitRPM	RPM less than this value will be set to the stallLimit, RPM values greater than limitRPM will scale linearly to freeLimit

3.6.2.41 SetSoftLimit()

```
CANError CANSparkMax::SetSoftLimit (
            CANSparkMax::SoftLimitDirection direction,
            double limit )
```

Set the soft limit based on position. The default unit is rotations, but will match the unit scaling set by the user.

Note that this value is not scaled internally so care must be taken to make sure these units match the desired conversion

Parameters

direction	the direction of motion to restrict
limit	position soft limit of the controller

3.6.2.42 StopMotor()

```
void CANSparkMax::StopMotor ( )  [override]
```

Common interface to stop the motor until Set is called again.

The documentation for this class was generated from the following files:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANSparkMax.h
- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/cpp/CANSparkMax.cpp

3.7 rev::CANSparkMaxLowLevel Class Reference

Inherits ErrorBase, and SpeedController.

Inherited by rev::CANSparkMax.

Classes

- struct FollowConfig
- struct PeriodicStatus0
- struct PeriodicStatus1
- struct PeriodicStatus2

Public Types

- enum **MotorType** { **kBrushed** = 0, **kBrushless** = 1 }
- enum **ParameterStatus** {
  kOK = 0, **kInvalidID** = 1, **kMismatchType** = 2, **kAccessMode** = 3,
  kInvalid = 4, **kNotImplementedDeprecated** = 5 }
- enum **PeriodicFrame** { **kStatus0** = 0, **kStatus1** = 1, **kStatus2** = 2 }

Public Member Functions

- CANSparkMaxLowLevel (int deviceID, MotorType type)
- ~CANSparkMaxLowLevel ()
- uint32_t GetFirmwareVersion ()
- uint32_t **GetFirmwareVersion** (bool &isDebugBuild)
- std::string GetFirmwareString ()
- std::vector< uint8_t > GetSerialNumber ()
- int GetDeviceId () const
- MotorType GetInitialMotorType ()
- CANError SetMotorType (MotorType type)
- MotorType GetMotorType ()
- CANError SetPeriodicFramePeriod (PeriodicFrame frame, int periodMs)
- void SetControlFramePeriodMs (int periodMs)
- CANError RestoreFactoryDefaults (bool persist=false)

Static Public Member Functions

- static void EnableExternalUSBControl (bool enable)
- static void SetEnable (bool enable)

Static Public Attributes

- static const uint8_t **kAPIMajorVersion**
- static const uint8_t **kAPIMinorVersion**
- static const uint8_t **kAPIBuildVersion**
- static const uint32_t **kAPIVersion**

Protected Member Functions

- CANError **SetEncPosition** (double value)
- CANError **SetIAccum** (double value)
- PeriodicStatus0 **GetPeriodicStatus0** ()
- PeriodicStatus1 **GetPeriodicStatus1** ()
- PeriodicStatus2 **GetPeriodicStatus2** ()
- CANError **SetFollow** (FollowConfig config)
- CANError **FollowerInvert** (bool invert)
- CANError **SetpointCommand** (double value, ControlType ctrl=ControlType::kDutyCycle, int pidSlot=0, double arbFeedforward=0, int arbFFUnits=0)
- float **GetSafeFloat** (float f)

Protected Attributes

- void * **m_sparkMax**
- MotorType **m_motorType**

Friends

- class **CANPIDController**
- class **CANDigitalInput**
- class **CANEncoder**
- class **CANAnalog**

3.7.1 Constructor & Destructor Documentation

3.7.1.1 CANSparkMaxLowLevel()

```
rev::CANSparkMaxLowLevel::CANSparkMaxLowLevel (
            int deviceID,
            MotorType type )  [explicit]
```

Create a new SPARK MAX Controller

Parameters

deviceID	The device ID.
type	The motor type connected to the controller. Brushless motors must be connected to their matching color and the hall sensor plugged in. Brushed motors must be connected to the Red and Black terminals only.

3.7.1.2 ~CANSparkMaxLowLevel()

```
rev::CANSparkMaxLowLevel::~CANSparkMaxLowLevel ( )
```

Closes the SPARK MAX Controller

3.7.2 Member Function Documentation

3.7.2.1 EnableExternalUSBControl()

```
static void rev::CANSparkMaxLowLevel::EnableExternalUSBControl (
            bool enable )  [static]
```

Allow external controllers to recieve control commands over USB. For example, a configuration where the heartbeat (and enable/disable) is sent by the main controller, but control frames are sent by other CAN devices over USB.

This is global for all controllers on the same bus.

This does not disable sending control frames from this device. To prevent conflicts, do not enable this feature and also send Set() for SetReference() from the controllers you wish to control.

Parameters

enable	Enable or disable external control

3.7.2.2 GetDeviceId()

```
int rev::CANSparkMaxLowLevel::GetDeviceId ( ) const
```

Get the configured Device ID of the SPARK MAX.

Returns

int device ID

3.7.2.3 GetFirmwareString()

```
std::string rev::CANSparkMaxLowLevel::GetFirmwareString ( )
```

Get the firmware version of the SPARK MAX as a string.

Returns

std::string Human readable firmware version string

3.7.2.4 GetFirmwareVersion()

```
uint32_t rev::CANSparkMaxLowLevel::GetFirmwareVersion ( )
```

Get the firmware version of the SPARK MAX.

Returns

uint32_t Firmware version integer. Value is represented as 4 bytes, Major.Minor.Build H.Build L

3.7.2.5 GetInitialMotorType()

```
MotorType rev::CANSparkMaxLowLevel::GetInitialMotorType ( )
```

Get the motor type setting from when the SparkMax was created.

This does not use the Get Parameter API which means it does not read what motor type is stored on the SparkMax itself. Instead, it reads the stored motor type from when the SparkMax object was first created.

Returns

MotorType Motor type setting

3.7.2.6 GetMotorType()

```
MotorType rev::CANSparkMaxLowLevel::GetMotorType ( )
```

Get the motor type setting for the SPARK MAX.

This uses the Get Parameter API and should be used infrequently. This function uses a non-blocking call and will return a cached value if the parameter is not returned by the timeout. The timeout can be changed by calling SetCANTimeout(int milliseconds)

Returns

MotorType Motor type setting

3.7.2.7 GetSerialNumber()

```
std::vector<uint8_t> rev::CANSparkMaxLowLevel::GetSerialNumber ( )
```

Get the unique serial number of the SPARK MAX. Currently not implemented.

Returns

std::vector<uint8_t> Vector of bytes representig the unique serial number

3.7.2.8 RestoreFactoryDefaults()

```
CANError rev::CANSparkMaxLowLevel::RestoreFactoryDefaults (
            bool persist = false )
```

Restore motor controller parameters to factory default

Parameters

persist	If true, burn the flash with the factory default parameters

Returns

CANError Set to CANError::kOk if successful

3.7.2.9 SetControlFramePeriodMs()

```
void rev::CANSparkMaxLowLevel::SetControlFramePeriodMs (
            int periodMs )
```

Set the control frame send period for the native CAN Send thread.

Parameters

periodMs	The send period in milliseconds between 1ms and 100ms

3.7.2.10 SetEnable()

```
static void rev::CANSparkMaxLowLevel::SetEnable (
            bool enable )  [static]
```

Send enabled or disabled command to controllers. This is global for all controllers on the same bus, and will only work for non-roboRIO targets in non-competiton use. This function will also not work if a roboRIO is present on the CAN bus.

This does not disable sending control frames from this device. To prevent conflicts, do not enable this feature and also send Set() for SetReference() from the controllers you wish to control.

Parameters

enable	Enable or disable external control

3.7.2.11 SetMotorType()

```
CANError rev::CANSparkMaxLowLevel::SetMotorType (
            MotorType type )
```

Set the motor type connected to the SPARK MAX.

This uses the Set Parameter API and should be used infrequently. The parameter does not presist unless burn↩ Flash() is called. The recommended method to configure this parameter is to use the SPARK MAX GUI to tune and save parameters.

Parameters

type	The type of motor connected to the controller. Brushless motors must be connected to their matching color and the hall sensor plugged in. Brushed motors must be connected to the Red and Black terminals only.

Returns

CANError Set to CANError::kOk if successful

3.7.2.12 SetPeriodicFramePeriod()

```
CANError rev::CANSparkMaxLowLevel::SetPeriodicFramePeriod (
            PeriodicFrame frame,
            int periodMs )
```

Set the rate of transmission for periodic frames from the SPARK MAX

Each motor controller sends back three status frames with different data at set rates. Use this function to change the default rates.

Defaults: Status0 - 10ms Status1 - 20ms Status2 - 50ms

This value is not stored in the FLASH after calling burnFlash() and is reset on powerup.

Refer to the SPARK MAX reference manual on details for how and when to configure this parameter.

Parameters

frameID	The frame ID can be one of PeriodicFrame type
periodMs	The rate the controller sends the frame to the controller.

Returns

CANError Set to CANError::kOk if successful

The documentation for this class was generated from the following file:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANSparkMaxLowLevel.h

3.8 rev::CANSparkMax::ExternalFollower Struct Reference

Public Attributes

- int **arbId**
- int **configId**

The documentation for this struct was generated from the following file:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANSparkMax.h

3.9 rev::CANSparkMaxLowLevel::FollowConfig Struct Reference

Public Attributes

- uint32_t **leaderArbId**
- union {
  struct {
  uint32_t **rsvd1**: 18
  uint32_t **invert**: 1
  uint32_t **rsvd2**: 5
  uint32_t **predefined**: 8
  } **config**
  uint32_t **configRaw**
  };

The documentation for this struct was generated from the following file:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANSparkMaxLowLevel.h

3.10 rev::CANSparkMaxLowLevel::PeriodicStatus0 Struct Reference

Public Attributes

- double **appliedOutput**
- uint16_t **faults**
- uint16_t **stickyFaults**
- MotorType **motorType**
- bool **isFollower**
- uint8_t **lock**
- uint8_t **roboRIO**
- uint8_t **isInverted**
- uint64_t **timestamp**

The documentation for this struct was generated from the following file:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANSparkMaxLowLevel.h

3.11 rev::CANSparkMaxLowLevel::PeriodicStatus1 Struct Reference

Public Attributes

- double **sensorVelocity**
- uint8_t **motorTemperature**
- double **busVoltage**
- double **outputCurrent**
- uint64_t **timestamp**

The documentation for this struct was generated from the following file:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANSparkMaxLowLevel.h

3.12 rev::CANSparkMaxLowLevel::PeriodicStatus2 Struct Reference

Public Attributes

- double **sensorPosition**
- double **iAccum**
- uint64_t **timestamp**

The documentation for this struct was generated from the following file:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/CANSparkMaxLowLevel.h

3.13 rev::SparkMax Class Reference

```
#include <SparkMax.h>
```

Inherits PWMSpeedController.

Public Member Functions

- SparkMax (int channel)
- **SparkMax** (SparkMax &&)=default
- SparkMax & **operator=** (SparkMax &&)=default

3.13.1 Detailed Description

REV Robotics CAN speed controller controlled via PWM.

3.13.2 Constructor & Destructor Documentation

3.13.2.1 SparkMax()

```
SparkMax::SparkMax (
            int channel )  [explicit]
```

Constructor for a Spark Max.

Parameters

channel	The PWM channel that the Spark is attached to. 0-9 are on-board, 10-19 are on the MXP port

The documentation for this class was generated from the following files:

- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/include/rev/SparkMax.h
- C:/Users/Will/Src/SPARK-MAX-roboRIO/src/main/native/cpp/SparkMax.cpp

Index