WPILib C Reference Manual

Version 1.0

Generated by Doxygen 1.5.7.1

Thu Jan 22 10:16:24 2009

Contents

Chapter 1

Deprecated List

Global GetEncoder Use GetEncoderDistance() in favor of this method. This returns unscaled pulses and GetDistance() scales using value from SetEncoderDistancePerPulse().

Global GetEncoder Use GetEncoderDistance() in favor of this method. This returns unscaled pulses and GetDistance() scales using value from SetEncoderDistancePerPulse().

Global GetEncoderPeriod Use GetEncoderRate() in favor of this method. This returns unscaled periods and GetEncoderRate() scales using value from SetEncoderDistancePerPulse().

Global GetEncoderPeriod Use GetEncoderRate() in favor of this method. This returns unscaled periods and GetEncoderRate() scales using value from SetEncoderDistancePerPulse().

Global SetMaxEncoderPeriod Use SetEncoderMinRate() in favor of this method. This takes unscaled periods and SetMinEncoderRate() scales using value from SetEncoderDistancePerPulse().

Global SetMaxEncoderPeriod Use SetEncoderMinRate() in favor of this method. This takes unscaled periods and SetMinEncoderRate() scales using value from SetEncoderDistancePerPulse().

Chapter 2

Bug List

Global PrintfSerial All pointer-based parameters seem to return an error.

Global ScanfSerial All pointer-based parameters seem to return an error.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 SimpleCRobot Class Reference

`#include <SimpleCRobot.h>`

Public Member Functions

- SimpleCRobot ()
- virtual ~SimpleCRobot ()
- void StartCompetition ()

5.1.1 Constructor & Destructor Documentation

5.1.1.1 SimpleCRobot::SimpleCRobot ()

The simple robot constructor. The constructor, besides doing the normal constructor stuff, also calls the Initialize() C function where sensors can be set up immediately after the power is turned on.

5.1.1.2 virtual SimpleCRobot::~SimpleCRobot () `[inline, virtual]`

5.1.2 Member Function Documentation

5.1.2.1 void SimpleCRobot::StartCompetition ()

Start a competition. This code needs to track the order of the field starting to ensure that everything happens in the right order. Repeatedly run the correct method, either Autonomous or OperatorControl when the robot is enabled. After running the correct method, wait for some state to change, either the other mode starts or the robot is disabled. Then go back and wait for the robot to be enabled again.

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

- SimpleCRobot.h
- SimpleCRobot.cpp

Chapter 6

File Documentation

6.1 CAccelerometer.cpp File Reference

```
#include "Accelerometer.h"
#include "CAccelerometer.h"
#include "AnalogModule.h"
#include "CWrappers.h"
```

Functions

- static Accelerometer * AllocateAccelerometer (UINT32 slot, UINT32 channel)
- float GetAcceleration (UINT32 channel)
- float GetAcceleration (UINT32 slot, UINT32 channel)
- void SetAccelerometerSensitivity (UINT32 channel, float sensitivity)
- void SetAccelerometerSensitivity (UINT32 slot, UINT32 channel, float sensitivity)
- void SetAccelerometerZero (UINT32 channel, float zero)
- void SetAccelerometerZero (UINT32 slot, UINT32 channel, float zero)
- void DeleteAccelerometer (UINT32 slot, UINT32 channel)
- void DeleteAccelerometer (UINT32 channel)

Variables

- static Accelerometer * accelerometers [SensorBase::kAnalogModules][SensorBase::kAnalogChannels]
- static bool initialized = false

6.1.1 Function Documentation

6.1.1.1 static Accelerometer* AllocateAccelerometer (UINT32 *slot*, UINT32 *channel*) [static]

Allocate an instance of the C Accelerometer object

Parameters:

slot The slot the analog module is plugged into

channel The analog module channel the accelerometer is plugged into

6.1.1.2 void DeleteAccelerometer (UINT32 *channel*)

Delete the accelerometer underlying object Deletes the object that is associated with this accelerometer and frees up the storage and the ports.

Parameters:

channel The channel the accelerometer is plugged into

6.1.1.3 void DeleteAccelerometer (UINT32 *slot*, UINT32 *channel*)

Delete the accelerometer underlying object Deletes the object that is associated with this accelerometer and frees up the storage and the ports.

Parameters:

slot The slot the analog module is plugged into

channel The channel the accelerometer is plugged into

6.1.1.4 float GetAcceleration (UINT32 *slot*, UINT32 *channel*)

Get the acceleration in Gs

Parameters:

slot The slot the analog module is plugged into

channel The channel the accelerometer is plugged into

Returns:

Returns the acceleration in Gs

6.1.1.5 float GetAcceleration (UINT32 *channel*)

Get the acceleration in Gs

Parameters:

channel The channel the accelerometer is plugged into

Returns:

Returns the acceleration in Gs

6.1.1.6 void SetAccelerometerSensitivity (UINT32 *slot*, UINT32 *channel*, float *sensitivity*)

Set the accelerometer sensitivity.

This sets the sensitivity of the accelerometer used for calculating the acceleration. The sensitivity varys by accelerometer model. There are constants defined for various models.

Parameters:

slot The slot the analog module is plugged into

channel The channel the accelerometer is plugged into

sensitivity The sensitivity of accelerometer in Volts per G.

6.1.1.7 void SetAccelerometerSensitivity (UINT32 *channel*, float *sensitivity*)

Set the accelerometer sensitivity.

This sets the sensitivity of the accelerometer used for calculating the acceleration. The sensitivity varys by accelerometer model. There are constants defined for various models.

Parameters:

channel The channel the accelerometer is plugged into

sensitivity The sensitivity of accelerometer in Volts per G.

6.1.1.8 void SetAccelerometerZero (UINT32 *slot*, UINT32 *channel*, float *zero*)

Set the voltage that corresponds to 0 G.

The zero G voltage varys by accelerometer model. There are constants defined for various models.

Parameters:

slot The slot the analog module is plugged into

channel The channel the accelerometer is plugged into

zero The zero G voltage.

6.1.1.9 void SetAccelerometerZero (UINT32 *channel*, float *zero*)

Set the voltage that corresponds to 0 G.

The zero G voltage varys by accelerometer model. There are constants defined for various models.

Parameters:

channel The channel the accelerometer is plugged into

zero The zero G voltage.

6.1.2 Variable Documentation

6.1.2.1 Accelerometer* accelerometers[SensorBase::kAnalogModules][SensorBase::kAnalogChannels] [static]

6.1.2.2 bool initialized = false [static]

6.2 CAccelerometer.h File Reference

Functions

- float GetAcceleration (UINT32 channel)
- float GetAcceleration (UINT32 slot, UINT32 channel)
- void SetAccelerometerSensitivity (UINT32 channel, float sensitivity)
- void SetAccelerometerSensitivity (UINT32 slot, UINT32 channel, float sensitivity)
- void SetAccelerometerZero (UINT32 channel, float zero)
- void SetAccelerometerZero (UINT32 slot, UINT32 channel, float zero)
- void DeleteAccelerometer (UINT32 slot, UINT32 channel)
- void DeleteAccelerometer (UINT32 channe)

6.2.1 Function Documentation

6.2.1.1 void DeleteAccelerometer (UINT32 *channel*)

Delete the accelerometer underlying object Deletes the object that is associated with this accelerometer and frees up the storage and the ports.

Parameters:

channel The channel the accelerometer is plugged into

6.2.1.2 void DeleteAccelerometer (UINT32 *slot*, UINT32 *channel*)

Delete the accelerometer underlying object Deletes the object that is associated with this accelerometer and frees up the storage and the ports.

Parameters:

slot The slot the analog module is plugged into

channel The channel the accelerometer is plugged into

6.2.1.3 float GetAcceleration (UINT32 *slot*, UINT32 *channel*)

Get the acceleration in Gs

Parameters:

slot The slot the analog module is plugged into

channel The channel the accelerometer is plugged into

Returns:

Returns the acceleration in Gs

6.2.1.4 float GetAcceleration (UINT32 *channel*)

Get the acceleration in Gs

Parameters:

channel The channel the accelerometer is plugged into

Returns:

Returns the acceleration in Gs

6.2.1.5 void SetAccelerometerSensitivity (UINT32 *slot*, UINT32 *channel*, float *sensitivity*)

Set the accelerometer sensitivity.

This sets the sensitivity of the accelerometer used for calculating the acceleration. The sensitivity varys by accelerometer model. There are constants defined for various models.

Parameters:

slot The slot the analog module is plugged into

channel The channel the accelerometer is plugged into

sensitivity The sensitivity of accelerometer in Volts per G.

6.2.1.6 void SetAccelerometerSensitivity (UINT32 *channel*, float *sensitivity*)

Set the accelerometer sensitivity.

This sets the sensitivity of the accelerometer used for calculating the acceleration. The sensitivity varys by accelerometer model. There are constants defined for various models.

Parameters:

channel The channel the accelerometer is plugged into

sensitivity The sensitivity of accelerometer in Volts per G.

6.2.1.7 void SetAccelerometerZero (UINT32 *slot*, UINT32 *channel*, float *zero*)

Set the voltage that corresponds to 0 G.

The zero G voltage varys by accelerometer model. There are constants defined for various models.

Parameters:

slot The slot the analog module is plugged into

channel The channel the accelerometer is plugged into

zero The zero G voltage.

6.2.1.8 void SetAccelerometerZero (UINT32 *channel*, float *zero*)

Set the voltage that corresponds to 0 G.

The zero G voltage varys by accelerometer model. There are constants defined for various models.

Parameters:

channel The channel the accelerometer is plugged into

zero The zero G voltage.

6.3 CAnalogChannel.cpp File Reference

```
#include "CAnalogChannel.h"
#include "AnalogModule.h"
```

Functions

- AnalogChannel * AllocateAnalogChannel (UINT32 slot, UINT32 channel)
- INT16 GetAnalogValue (UINT32 slot, UINT32 channel)
- INT32 GetAnalogAverageValue (UINT32 slot, UINT32 channel)
- float GetAnalogVoltage (UINT32 slot, UINT32 channel)
- float GetAnalogAverageVoltage (UINT32 slot, UINT32 channel)
- void SetAnalogAverageBits (UINT32 slot, UINT32 channel, UINT32 bits)
- UINT32 GetAnalogAverageBits (UINT32 slot, UINT32 channel)
- void SetAnalogOversampleBits (UINT32 slot, UINT32 channel, UINT32 bits)
- UINT32 GetAnalogOversampleBits (UINT32 slot, UINT32 channel)
- INT16 GetAnalogValue (UINT32 channel)
- INT32 GetAnalogAverageValue (UINT32 channel)
- float GetAnalogVoltage (UINT32 channel)
- float GetAnalogAverageVoltage (UINT32 channel)
- void SetAnalogAverageBits (UINT32 channel, UINT32 bits)
- UINT32 GetAnalogAverageBits (UINT32 channel)
- void SetAnalogOversampleBits (UINT32 channel, UINT32 bits)
- UINT32 GetAnalogOversampleBits (UINT32 channel)
- void DeleteAnalogChannel (UINT32 slot, UINT32 channel)
- void DeleteAnalogChannel (UINT32 channel)

Variables

- static bool analogChannelsInitialized = false
- static AnalogChannel * analogs [SensorBase::kAnalogModules][SensorBase::kAnalogChannels]

6.3.1 Function Documentation

6.3.1.1 AnalogChannel* AllocateAnalogChannel (UINT32 *slot*, UINT32 *channel*)

Allocate an AnalogChannel object for this set of slot/port

Parameters:

slot The slot the analog module is plugged into

channel The channel number on the module for this analog channel object

6.3.1.2 void DeleteAnalogChannel (UINT32 *channel*)

Delete the resources associated with this AnalogChannel The underlying object and the port reservations are deleted for this analog channel.

Parameters:

channel The channel in the module assicated with this analog channel

6.3.1.3 void DeleteAnalogChannel (UINT32 *slot*, UINT32 *channel*)

Delete the resources associated with this AnalogChannel The underlying object and the port reservations are deleted for this analog channel.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

6.3.1.4 UINT32 GetAnalogAverageBits (UINT32 *channel*)

Get the number of averaging bits previously configured. This gets the number of averaging bits from the FPGA. The actual number of averaged samples is 2∗∗bits. The averaging is done automatically in the FPGA.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

Number of bits of averaging previously configured.

6.3.1.5 UINT32 GetAnalogAverageBits (UINT32 *slot*, UINT32 *channel*)

Get the number of averaging bits previously configured. This gets the number of averaging bits from the FPGA. The actual number of averaged samples is 2∗∗bits. The averaging is done automatically in the FPGA.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

Returns:

Number of bits of averaging previously configured.

6.3.1.6 INT32 GetAnalogAverageValue (UINT32 *channel*)

Get a sample from the output of the oversample and average engine for this channel. The sample is 12-bit + the value configured in SetOversampleBits(). The value configured in SetAverageBits() will cause this value to be averaged 2**bits number of samples. This is not a sliding window. The sample will not change until 2**(OversamplBits + AverageBits) samples have been acquired from the module on this channel. Use GetAverageVoltage() to get the analog value in calibrated units.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

A sample from the oversample and average engine for this channel.

6.3.1.7 INT32 GetAnalogAverageValue (UINT32 *slot*, UINT32 *channel*)

Get a sample from the output of the oversample and average engine for this channel. The sample is 12-bit + the value configured in SetOversampleBits(). The value configured in SetAverageBits() will cause this value to be averaged 2**bits number of samples. This is not a sliding window. The sample will not change until 2**(OversamplBits + AverageBits) samples have been acquired from the module on this channel. Use GetAverageVoltage() to get the analog value in calibrated units.

Parameters:

slot The slot the analog module is plugged into

channel the channel for the value being used

Returns:

A sample from the oversample and average engine for this channel.

6.3.1.8 float GetAnalogAverageVoltage (UINT32 *channel*)

Get a scaled sample from the output of the oversample and average engine for this channel. The value is scaled to units of Volts using the calibrated scaling data from GetLSBWeight() and GetOffset(). Using oversampling will cause this value to be higher resolution, but it will update more slowly. Using averaging will cause this value to be more stable, but it will update more slowly.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

A scaled sample from the output of the oversample and average engine for this channel.

6.3.1.9 float GetAnalogAverageVoltage (UINT32 *slot*, UINT32 *channel*)

Get a scaled sample from the output of the oversample and average engine for this channel. The value is scaled to units of Volts using the calibrated scaling data from GetLSBWeight() and GetOffset(). Using oversampling will cause this value to be higher resolution, but it will update more slowly. Using averaging will cause this value to be more stable, but it will update more slowly.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

Returns:

A scaled sample from the output of the oversample and average engine for this channel.

6.3.1.10 UINT32 GetAnalogOversampleBits (UINT32 *channel*)

Get the number of oversample bits previously configured. This gets the number of oversample bits from the FPGA. The actual number of oversampled values is 2∗∗bits. The oversampling is done automatically in the FPGA.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

Number of bits of oversampling previously configured.

6.3.1.11 UINT32 GetAnalogOversampleBits (UINT32 *slot*, UINT32 *channel*)

Get the number of oversample bits previously configured. This gets the number of oversample bits from the FPGA. The actual number of oversampled values is 2∗∗bits. The oversampling is done automatically in the FPGA.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

Returns:

Number of bits of oversampling previously configured.

6.3.1.12 INT16 GetAnalogValue (UINT32 *channel*)

Get a sample straight from this channel on the module. The sample is a 12-bit value representing the -10V to 10V range of the A/D converter in the module. The units are in A/D converter codes. Use GetVoltage() to get the analog value in calibrated units.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

A sample straight from this channel on the module.

6.3.1.13 INT16 GetAnalogValue (UINT32 *slot*, UINT32 *channel*)

Get a sample straight from this channel on the module. The sample is a 12-bit value representing the -10V to 10V range of the A/D converter in the module. The units are in A/D converter codes. Use GetVoltage() to get the analog value in calibrated units.

Parameters:

slot The slot the analog module is plugged into

channel the channel for the value being used

Returns:

A sample straight from this channel on the module.

6.3.1.14 float GetAnalogVoltage (UINT32 *channel*)

Get a scaled sample straight from this channel on the module. The value is scaled to units of Volts using the calibrated scaling data from GetLSBWeight() and GetOffset().

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

A scaled sample straight from this channel on the module.

6.3.1.15 float GetAnalogVoltage (UINT32 *slot*, UINT32 *channel*)

Get a scaled sample straight from this channel on the module. The value is scaled to units of Volts using the calibrated scaling data from GetLSBWeight() and GetOffset().

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

Returns:

A scaled sample straight from this channel on the module.

6.3.1.16 void SetAnalogAverageBits (UINT32 *channel*, UINT32 *bits*)

Set the number of averaging bits. This sets the number of averaging bits. The actual number of averaged samples is 2**bits. Use averaging to improve the stability of your measurement at the expense of sampling rate. The averaging is done automatically in the FPGA.

Parameters:

channel The channel in the module assicated with this analog channel

bits Number of bits of averaging.

6.3.1.17 void SetAnalogAverageBits (UINT32 *slot*, UINT32 *channel*, UINT32 *bits*)

Set the number of averaging bits. This sets the number of averaging bits. The actual number of averaged samples is 2**bits. Use averaging to improve the stability of your measurement at the expense of sampling rate. The averaging is done automatically in the FPGA.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

bits Number of bits of averaging.

6.3.1.18 void SetAnalogOversampleBits (UINT32 *channel*, UINT32 *bits*)

Set the number of oversample bits. This sets the number of oversample bits. The actual number of oversampled values is 2**bits. Use oversampling to improve the resolution of your measurements at the expense of sampling rate. The oversampling is done automatically in the FPGA.

Parameters:

channel The channel in the module assicated with this analog channel

bits Number of bits of oversampling.

6.3.1.19 void SetAnalogOversampleBits (UINT32 *slot*, UINT32 *channel*, UINT32 *bits*)

Set the number of oversample bits. This sets the number of oversample bits. The actual number of oversampled values is 2**bits. Use oversampling to improve the resolution of your measurements at the expense of sampling rate. The oversampling is done automatically in the FPGA.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

bits Number of bits of oversampling.

6.3.2 Variable Documentation

6.3.2.1 bool analogChannelsInitialized = false [static]

6.3.2.2 AnalogChannel* analogs[SensorBase::kAnalogModules][SensorBase::kAnalogChannels] [static]

6.4 CAnalogChannel.h File Reference

```
#include "AnalogChannel.h"
#include "CWrappers.h"
```

Functions

- AnalogChannel * AllocateAnalogChannel (UINT32 module, UINT32 channel)
- INT16 GetAnalogValue (UINT32 slot, UINT32 channel)
- INT32 GetAnalogAverageValue (UINT32 slot, UINT32 channel)
- float GetAnalogVoltage (UINT32 slot, UINT32 channel)
- float GetAnalogAverageVoltage (UINT32 slot, UINT32 channel)
- void SetAnalogAverageBits (UINT32 slot, UINT32 channel, UINT32 bits)
- UINT32 GetAnalogAverageBits (UINT32 slot, UINT32 channel)
- void SetAnalogOversampleBits (UINT32 slot, UINT32 slot, UINT32 channel, UINT32 bits)
- UINT32 GetAnalogOversampleBits (UINT32 channel)
- INT16 GetAnalogValue (UINT32 channel)
- INT32 GetAnalogAverageValue (UINT32 channel)
- float GetAnalogVoltage (UINT32 channel)
- float GetAnalogAverageVoltage (UINT32 channel)
- void SetAnalogAverageBits (UINT32 channel, UINT32 bits)
- UINT32 GetAnalogAverageBits (UINT32 channel)
- void SetAnalogOversampleBits (UINT32 channel, UINT32 bits)
- UINT32 GetAnalogLSBWeight ()
- INT32 GetAnalogOffset ()
- void DeleteAnalogChannel (UINT32 slot, UINT32 channel)
- void DeleteAnalogChannel (UINT32 channel)

6.4.1 Function Documentation

6.4.1.1 AnalogChannel* AllocateAnalogChannel (UINT32 *slot*, UINT32 *channel*)

Allocate an AnalogChannel object for this set of slot/port

Parameters:

slot The slot the analog module is plugged into

channel The channel number on the module for this analog channel object

6.4.1.2 void DeleteAnalogChannel (UINT32 *channel*)

Delete the resources associated with this AnalogChannel The underlying object and the port reservations are deleted for this analog channel.

Parameters:

channel The channel in the module assicated with this analog channel

6.4.1.3 void DeleteAnalogChannel (UINT32 *slot*, UINT32 *channel*)

Delete the resources associated with this AnalogChannel The underlying object and the port reservations are deleted for this analog channel.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

6.4.1.4 UINT32 GetAnalogAverageBits (UINT32 *channel*)

Get the number of averaging bits previously configured. This gets the number of averaging bits from the FPGA. The actual number of averaged samples is 2∗∗bits. The averaging is done automatically in the FPGA.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

Number of bits of averaging previously configured.

6.4.1.5 UINT32 GetAnalogAverageBits (UINT32 *slot*, UINT32 *channel*)

Get the number of averaging bits previously configured. This gets the number of averaging bits from the FPGA. The actual number of averaged samples is 2∗∗bits. The averaging is done automatically in the FPGA.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

Returns:

Number of bits of averaging previously configured.

6.4.1.6 INT32 GetAnalogAverageValue (UINT32 *channel*)

Get a sample from the output of the oversample and average engine for this channel. The sample is 12-bit + the value configured in SetOversampleBits(). The value configured in SetAverageBits() will cause this value to be averaged 2∗∗bits number of samples. This is not a sliding window. The sample will not change until 2∗∗(OversamplBits + AverageBits) samples have been acquired from the module on this channel. Use GetAverageVoltage() to get the analog value in calibrated units.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

A sample from the oversample and average engine for this channel.

6.4.1.7 INT32 GetAnalogAverageValue (UINT32 *slot*, UINT32 *channel*)

Get a sample from the output of the oversample and average engine for this channel. The sample is 12-bit + the value configured in SetOversampleBits(). The value configured in SetAverageBits() will cause this value to be averaged 2**bits number of samples. This is not a sliding window. The sample will not change until 2**(OversamplBits + AverageBits) samples have been acquired from the module on this channel. Use GetAverageVoltage() to get the analog value in calibrated units.

Parameters:

slot The slot the analog module is plugged into

channel the channel for the value being used

Returns:

A sample from the oversample and average engine for this channel.

6.4.1.8 float GetAnalogAverageVoltage (UINT32 *channel*)

Get a scaled sample from the output of the oversample and average engine for this channel. The value is scaled to units of Volts using the calibrated scaling data from GetLSBWeight() and GetOffset(). Using oversampling will cause this value to be higher resolution, but it will update more slowly. Using averaging will cause this value to be more stable, but it will update more slowly.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

A scaled sample from the output of the oversample and average engine for this channel.

6.4.1.9 float GetAnalogAverageVoltage (UINT32 *slot*, UINT32 *channel*)

Get a scaled sample from the output of the oversample and average engine for this channel. The value is scaled to units of Volts using the calibrated scaling data from GetLSBWeight() and GetOffset(). Using oversampling will cause this value to be higher resolution, but it will update more slowly. Using averaging will cause this value to be more stable, but it will update more slowly.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

Returns:

A scaled sample from the output of the oversample and average engine for this channel.

6.4.1.10 UINT32 GetAnalogLSBWeight ()

6.4.1.11 INT32 GetAnalogOffset ()

6.4.1.12 UINT32 GetAnalogOversampleBits (UINT32 *channel*)

Get the number of oversample bits previously configured. This gets the number of oversample bits from the FPGA. The actual number of oversampled values is 2∗∗bits. The oversampling is done automatically in the FPGA.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

Number of bits of oversampling previously configured.

6.4.1.13 INT16 GetAnalogValue (UINT32 *channel*)

Get a sample straight from this channel on the module. The sample is a 12-bit value representing the -10V to 10V range of the A/D converter in the module. The units are in A/D converter codes. Use GetVoltage() to get the analog value in calibrated units.

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

A sample straight from this channel on the module.

6.4.1.14 INT16 GetAnalogValue (UINT32 *slot*, UINT32 *channel*)

Get a sample straight from this channel on the module. The sample is a 12-bit value representing the -10V to 10V range of the A/D converter in the module. The units are in A/D converter codes. Use GetVoltage() to get the analog value in calibrated units.

Parameters:

slot The slot the analog module is plugged into

channel the channel for the value being used

Returns:

A sample straight from this channel on the module.

6.4.1.15 float GetAnalogVoltage (UINT32 *channel*)

Get a scaled sample straight from this channel on the module. The value is scaled to units of Volts using the calibrated scaling data from GetLSBWeight() and GetOffset().

Parameters:

channel The channel in the module assicated with this analog channel

Returns:

A scaled sample straight from this channel on the module.

6.4.1.16 float GetAnalogVoltage (UINT32 *slot*, UINT32 *channel*)

Get a scaled sample straight from this channel on the module. The value is scaled to units of Volts using the calibrated scaling data from GetLSBWeight() and GetOffset().

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

Returns:

A scaled sample straight from this channel on the module.

6.4.1.17 void SetAnalogAverageBits (UINT32 *channel*, UINT32 *bits*)

Set the number of averaging bits. This sets the number of averaging bits. The actual number of averaged samples is 2∗∗bits. Use averaging to improve the stability of your measurement at the expense of sampling rate. The averaging is done automatically in the FPGA.

Parameters:

channel The channel in the module assicated with this analog channel

bits Number of bits of averaging.

6.4.1.18 void SetAnalogAverageBits (UINT32 *slot*, UINT32 *channel*, UINT32 *bits*)

Set the number of averaging bits. This sets the number of averaging bits. The actual number of averaged samples is 2∗∗bits. Use averaging to improve the stability of your measurement at the expense of sampling rate. The averaging is done automatically in the FPGA.

Parameters:

slot The slot the analog module is plugged into

channel The channel in the module assicated with this analog channel

bits Number of bits of averaging.

6.4.1.19 void SetAnalogOversampleBits (UINT32 *channel*, UINT32 *bits*)

Set the number of oversample bits. This sets the number of oversample bits. The actual number of oversampled values is 2∗∗bits. Use oversampling to improve the resolution of your measurements at the expense of sampling rate. The oversampling is done automatically in the FPGA.

Parameters:

channel The channel in the module assicated with this analog channel

bits Number of bits of oversampling.

6.4.1.20 void SetAnalogOversampleBits (UINT32 *slot*, UINT32 *slot*, UINT32 *channel*, UINT32 *bits*)

6.5 CCompressor.cpp File Reference

```
#include "Compressor.h"
#include "CCompressor.h"
#include "Utility.h"
#include "WPIStatus.h"
```

Functions

- void CreateCompressor (UINT32 pressureSwitchChannel, UINT32 relayChannel)
- void CreateCompressor (UINT32 pressureSwitchSlot, UINT32 pressureSwitchChannel, UINT32 relaySlot, UINT32 relayChannel)
- void StartCompressor ()
- void StopCompressor ()
- bool CompressorEnabled ()
- void DeleteCompressor ()

Variables

- static Compressor * compressor = NULL

6.5.1 Function Documentation

6.5.1.1 bool CompressorEnabled ()

Get the state of the enabled flag. Return the state of the enabled flag for the compressor and pressure switch.

Returns:

The state of the compressor task's enable flag.

6.5.1.2 void CreateCompressor (UINT32 *pressureSwitchSlot*, UINT32 *pressureSwitchChannel*, UINT32 *relaySlot*, UINT32 *relayChannel*)

Allocate resources for a compressor/pressure switch pair Allocate the underlying object for the compressor.

Parameters:

pressureSwitchSlot The slot of the digital module for the pressure switch

pressureSwitchChannel The channel on the digital module for the pressure switch

relaySlot The slot of the digital module for the relay controlling the compressor

relayChannel The channel on the digital module for the relay that controls the compressor

6.5.1.3 void CreateCompressor (UINT32 *pressureSwitchChannel*, UINT32 *relayChannel*)

Allocate resources for a compressor/pressure switch pair Allocate the underlying object for the compressor.

Parameters:

pressureSwitchChannel The channel on the default digital module for the pressure switch

relayChannel The channel on the default digital module for the relay that controls the compressor

6.5.1.4 void DeleteCompressor ()

Free the resources associated with the compressor. The underlying Compressor object will be deleted and the resources and ports freed.

6.5.1.5 void StartCompressor ()

Start the compressor Calling this function will cause the compressor task to begin polling the switch and operating the compressor.

6.5.1.6 void StopCompressor ()

Stop the compressor. Stops the polling loop that operates the compressor. At this time the compressor will stop operating.

6.5.2 Variable Documentation

6.5.2.1 Compressor* compressor = NULL [static]

6.6 CCompressor.h File Reference

Functions

- void CreateCompressor (UINT32 pressureSwitch, UINT32 relayChannel)
- void CreateCompressor (UINT32 pressureSwitchSlot, UINT32 pressureSwitchChannel, UINT32 relaySlot, UINT32 relayChannel)
- void StartCompressor ()
- void StopCompressor ()
- bool CompressorEnabled ()
- void DeleteCompressor ()

6.6.1 Function Documentation

6.6.1.1 bool CompressorEnabled ()

Get the state of the enabled flag. Return the state of the enabled flag for the compressor and pressure switch.

Returns:

The state of the compressor task's enable flag.

6.6.1.2 void CreateCompressor (UINT32 *pressureSwitchSlot*, UINT32 *pressureSwitchChannel*, UINT32 *relaySlot*, UINT32 *relayChannel*)

Allocate resources for a compressor/pressure switch pair Allocate the underlying object for the compressor.

Parameters:

pressureSwitchSlot The slot of the digital module for the pressure switch

pressureSwitchChannel The channel on the digital module for the pressure switch

relaySlot The slot of the digital module for the relay controlling the compressor

relayChannel The channel on the digital module for the relay that controls the compressor

6.6.1.3 void CreateCompressor (UINT32 *pressureSwitchChannel*, UINT32 *relayChannel*)

Allocate resources for a compressor/pressure switch pair Allocate the underlying object for the compressor.

Parameters:

pressureSwitchChannel The channel on the default digital module for the pressure switch

relayChannel The channel on the default digital module for the relay that controls the compressor

6.6.1.4 void DeleteCompressor ()

Free the resources associated with the compressor. The underlying Compressor object will be deleted and the resources and ports freed.

6.6.1.5 void StartCompressor ()

Start the compressor Calling this function will cause the compressor task to begin polling the switch and operating the compressor.

6.6.1.6 void StopCompressor ()

Stop the compressor. Stops the polling loop that operates the compressor. At this time the compressor will stop operating.

6.7 CCounter.cpp File Reference

```
#include "VxWorks.h"
#include "CCounter.h"
#include "Counter.h"
#include "DigitalModule.h"
```

Functions

- static Counter * AllocateCounter (UINT32 slot, UINT32 channel)
- static Counter * AllocateCounter (UINT32 channel)
- void StartCounter (UINT32 slot, UINT32 channel)
- void StartCounter (UINT32 channel)
- INT32 GetCounter (UINT32 channel)
- INT32 GetCounter (UINT32 slot, UINT32 channel)
- void ResetCounter (UINT32 channel)
- void ResetCounter (UINT32 slot, UINT32 channel)
- void StopCounter (UINT32 slot, UINT32 channel)
- void StopCounter (UINT32 channel)
- double GetCounterPeriod (UINT32 slot, UINT32 channel)
- double GetCounterPeriod (UINT32 channel)
- void DeleteCounter (UINT32 slot, UINT32 channel)
- void DeleteCounter (UINT32 channel)

Variables

- static Counter * counters [SensorBase::kDigitalModules][SensorBase::kDigitalChannels]
- static bool initialized = false

6.7.1 Function Documentation

6.7.1.1 static Counter* AllocateCounter (UINT32 *channel*) [static]

Allocate the resource for a counter Allocate the underlying Counter object and the resources associated with the slot and channel

Parameters:

channel The channel of the digital input used with this counter

6.7.1.2 static Counter* AllocateCounter (UINT32 *slot*, UINT32 *channel*) [static]

Allocate the resource for a counter Allocate the underlying Counter object and the resources associated with the slot and channel

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.7.1.3 void DeleteCounter (UINT32 *channel*)

Delete the resources associated with this counter. The resources including the underlying object are deleted for this counter.

Parameters:

channel The channel of the digital input used with this counter

6.7.1.4 void DeleteCounter (UINT32 *slot*, UINT32 *channel*)

Delete the resources associated with this counter. The resources including the underlying object are deleted for this counter.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.7.1.5 INT32 GetCounter (UINT32 *slot*, UINT32 *channel*)

Read the current counter value. Read the value at this instant. It may still be running, so it reflects the current value. Next time it is read, it might have a different value.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.7.1.6 INT32 GetCounter (UINT32 *channel*)

Read the current counter value. Read the value at this instant. It may still be running, so it reflects the current value. Next time it is read, it might have a different value.

Parameters:

channel The channel of the digital input used with this counter

6.7.1.7 double GetCounterPeriod (UINT32 *channel*)

6.7.1.8 double GetCounterPeriod (UINT32 *slot*, UINT32 *channel*)

6.7.1.9 void ResetCounter (UINT32 *slot*, UINT32 *channel*)

Reset the Counter to zero. Set the counter value to zero. This doesn't effect the running state of the counter, just sets the current value to zero.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.7.1.10 void ResetCounter (UINT32 *channel*)

Reset the Counter to zero. Set the counter value to zero. This doesn't effect the running state of the counter, just sets the current value to zero.

Parameters:

channel The channel of the digital input used with this counter

6.7.1.11 void StartCounter (UINT32 *channel*)

Start the Counter counting. This enables the counter and it starts accumulating counts from the associated input channel. The counter value is not reset on starting, and still has the previous value.

Parameters:

channel The channel of the digital input used with this counter

6.7.1.12 void StartCounter (UINT32 *slot*, UINT32 *channel*)

Start the Counter counting. This enables the counter and it starts accumulating counts from the associated input channel. The counter value is not reset on starting, and still has the previous value.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.7.1.13 void StopCounter (UINT32 *channel*)

Stop the Counter. Stops the counting but doesn't effect the current value.

Parameters:

channel The channel of the digital input used with this counter

6.7.1.14 void StopCounter (UINT32 *slot*, UINT32 *channel*)

Stop the Counter. Stops the counting but doesn't effect the current value.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.7.2 Variable Documentation

6.7.2.1 Counter* counters[SensorBase::kDigitalModules][SensorBase::kDigitalChannels] [static]

6.7.2.2 bool initialized = false [static]

6.8 CCounter.h File Reference

Functions

- void StartCounter (UINT32 channel)
- void StartCounter (UINT32 slot, UINT32 channel)
- INT32 GetCounter (UINT32 channel)
- INT32 GetCounter (UINT32 slot, UINT32 channel)
- void ResetCounter (UINT32 channel)
- void ResetCounter (UINT32 slot, UINT32 channel)
- void StopCounter (UINT32 channel)
- void StopCounter (UINT32 slot, UINT32 channel)
- double GetCounterPeriod (UINT32 channel)
- double GetCounterPeriod (UINT32 slot, UINT32 channel)
- void DeleteCounter (UINT32 slot, UINT32 channel)
- void DeleteCounter (UINT32 channel)

6.8.1 Function Documentation

6.8.1.1 void DeleteCounter (UINT32 *channel*)

Delete the resources associated with this counter. The resources including the underlying object are deleted for this counter.

Parameters:

channel The channel of the digital input used with this counter

6.8.1.2 void DeleteCounter (UINT32 *slot*, UINT32 *channel*)

Delete the resources associated with this counter. The resources including the underlying object are deleted for this counter.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.8.1.3 INT32 GetCounter (UINT32 *slot*, UINT32 *channel*)

Read the current counter value. Read the value at this instant. It may still be running, so it reflects the current value. Next time it is read, it might have a different value.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.8.1.4 INT32 GetCounter (UINT32 *channel*)

Read the current counter value. Read the value at this instant. It may still be running, so it reflects the current value. Next time it is read, it might have a different value.

Parameters:

channel The channel of the digital input used with this counter

6.8.1.5 double GetCounterPeriod (UINT32 *slot*, UINT32 *channel*)

6.8.1.6 double GetCounterPeriod (UINT32 *channel*)

6.8.1.7 void ResetCounter (UINT32 *slot*, UINT32 *channel*)

Reset the Counter to zero. Set the counter value to zero. This doesn't effect the running state of the counter, just sets the current value to zero.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.8.1.8 void ResetCounter (UINT32 *channel*)

Reset the Counter to zero. Set the counter value to zero. This doesn't effect the running state of the counter, just sets the current value to zero.

Parameters:

channel The channel of the digital input used with this counter

6.8.1.9 void StartCounter (UINT32 *slot*, UINT32 *channel*)

Start the Counter counting. This enables the counter and it starts accumulating counts from the associated input channel. The counter value is not reset on starting, and still has the previous value.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.8.1.10 void StartCounter (UINT32 *channel*)

Start the Counter counting. This enables the counter and it starts accumulating counts from the associated input channel. The counter value is not reset on starting, and still has the previous value.

Parameters:

channel The channel of the digital input used with this counter

6.8.1.11 void StopCounter (UINT32 *slot*, UINT32 *channel*)

Stop the Counter. Stops the counting but doesn't effect the current value.

Parameters:

slot The slot the digital module is plugged into

channel The channel of the digital input used with this counter

6.8.1.12 void StopCounter (UINT32 *channel*)

Stop the Counter. Stops the counting but doesn't effect the current value.

Parameters:

channel The channel of the digital input used with this counter

6.9 CDigitalInput.cpp File Reference

```
#include "DigitalModule.h"
#include "DigitalInput.h"
#include "CDigitalInput.h"
```

Functions

- static DigitalInput * AllocateDigitalInput (UINT32 slot, UINT32 channel)
- UINT32 GetDigitalInput (UINT32 slot, UINT32 channel)
- UINT32 GetDigitalInput (UINT32 channel)
- void DeleteDigitalInput (UINT32 slot, UINT32 channel)
- void DeleteDigitalInput (UINT32 channel)

Variables

- static DigitalInput * digitalInputs [SensorBase::kDigitalModules][SensorBase::kDigitalChannels]
- static bool initialized = false

6.9.1 Function Documentation

6.9.1.1 static DigitalInput* AllocateDigitalInput (UINT32 *slot*, UINT32 *channel*) [static]

Allocates the resources associated with a DigitalInput. Allocate the underlying DigitalInput object and the reservations for the associated slot and channel.

Parameters:

slot The slot the digital input module is plugged into

channel The particular channel this digital input is using

6.9.1.2 void DeleteDigitalInput (UINT32 *channel*)

Frees the resources for this DigitalInput. Deletes the underlying object and frees the reservation for the associated digital input port.

Parameters:

channel The particular channel this digital input is using

6.9.1.3 void DeleteDigitalInput (UINT32 *slot*, UINT32 *channel*)

Frees the resources for this DigitalInput. Deletes the underlying object and frees the reservation for the associated digital input port.

Parameters:

slot The slot the digital input module is plugged into

channel The particular channel this digital input is using

6.9.1.4 UINT32 GetDigitalInput (UINT32 *channel*)

6.9.1.5 UINT32 GetDigitalInput (UINT32 *slot*, UINT32 *channel*)

6.9.2 Variable Documentation

6.9.2.1 DigitalInput* digitalInputs[SensorBase::kDigitalModules][SensorBase::kDigitalChannels] [static]

6.9.2.2 bool initialized = false [static]

6.10 CDigitalInput.h File Reference

Defines

- #define _C_DIGITIL_INPUT_H

Functions

- UINT32 GetDigitalInput (UINT32 slot, UINT32 channel)
- UINT32 GetDigitalInput (UINT32 channel)
- void DeleteDigitalInput (UINT32 slot, UINT32 channel)
- void DeleteDigitalInput (UINT32 channel)

6.10.1 Define Documentation

6.10.1.1 #define _C_DIGITIL_INPUT_H

6.10.2 Function Documentation

6.10.2.1 void DeleteDigitalInput (UINT32 *channel*)

Frees the resources for this DigitalInput. Deletes the underlying object and frees the reservation for the associated digital input port.

Parameters:

channel The particular channel this digital input is using

6.10.2.2 void DeleteDigitalInput (UINT32 *slot*, UINT32 *channel*)

Frees the resources for this DigitalInput. Deletes the underlying object and frees the reservation for the associated digital input port.

Parameters:

slot The slot the digital input module is plugged into

channel The particular channel this digital input is using

6.10.2.3 UINT32 GetDigitalInput (UINT32 *channel*)

6.10.2.4 UINT32 GetDigitalInput (UINT32 *slot*, UINT32 *channel*)

6.11 CDigitalOutput.cpp File Reference

```
#include "DigitalModule.h"
#include "DigitalOutput.h"
#include "CDigitalOutput.h"
```

Functions

- static DigitalOutput * AllocateDigitalOutput (UINT32 slot, UINT32 channel)
- void SetDigitalOutput (UINT32 slot, UINT32 channel, UINT32 value)
- void SetDigitalOutput (UINT32 channel, UINT32 value)
- void DeleteDigitalOutput (UINT32 slot, UINT32 channel)
- void DeleteDigitalOutput (UINT32 channel)

Variables

- static DigitalOutput * digitalOutputs [SensorBase::kDigitalModules][SensorBase::kDigitalChannels]
- static bool initialized = false

6.11.1 Function Documentation

6.11.1.1 static DigitalOutput* AllocateDigitalOutput (UINT32 *slot*, UINT32 *channel*) [static]

Allocate the DigitalOuput. Allocates the resources associated with this DigitalOutput including the channel/slot reservation and the underlying DigitalOutput object.

Parameters:

slot The slot this digital module is plugged into

channel The channel being used for this digital output

6.11.1.2 void DeleteDigitalOutput (UINT32 *channel*)

Free the resources associated with this digital output. The underlying DigitalOutput object and the resouces for the channel and slot are freed so they can be reused.

Parameters:

channel The channel being used for this digital output

6.11.1.3 void DeleteDigitalOutput (UINT32 *slot*, UINT32 *channel*)

Free the resources associated with this digital output. The underlying DigitalOutput object and the resouces for the channel and slot are freed so they can be reused.

Parameters:

slot The slot this digital module is plugged into

channel The channel being used for this digital output

6.11.1.4 void SetDigitalOutput (UINT32 *channel*, UINT32 *value*)

Set the value of a digital output. Set the value of a digital output to either one (true) or zero (false).

Parameters:

channel The channel being used for this digital output

value The 0/1 value set to the port.

6.11.1.5 void SetDigitalOutput (UINT32 *slot*, UINT32 *channel*, UINT32 *value*)

Set the value of a digital output. Set the value of a digital output to either one (true) or zero (false).

Parameters:

slot The slot this digital module is plugged into

channel The channel being used for this digital output

value The 0/1 value set to the port.

6.11.2 Variable Documentation

6.11.2.1 DigitalOutput* digitalOutputs[SensorBase::kDigitalModules][SensorBase::kDigitalChannels] [static]

6.11.2.2 bool initialized = false [static]

6.12 CDigitalOutput.h File Reference

Defines

- #define _C_DIGITIL_OUTPUT_H

Functions

- void SetDigitalOutput (UINT32 slot, UINT32 channel, UINT32 value)
- void SetDigitalOutput (UINT32 channel, UINT32 value)
- void DeleteDigitalOutput (UINT32 slot, UINT32 channel)
- void DeleteDigitalOutput (UINT32 channel)

6.12.1 Define Documentation

6.12.1.1 #define _C_DIGITIL_OUTPUT_H

6.12.2 Function Documentation

6.12.2.1 void DeleteDigitalOutput (UINT32 *channel*)

Free the resources associated with this digital output. The underlying DigitalOutput object and the resouces for the channel and slot are freed so they can be reused.

Parameters:

channel The channel being used for this digital output

6.12.2.2 void DeleteDigitalOutput (UINT32 *slot*, UINT32 *channel*)

Free the resources associated with this digital output. The underlying DigitalOutput object and the resouces for the channel and slot are freed so they can be reused.

Parameters:

slot The slot this digital module is plugged into

channel The channel being used for this digital output

6.12.2.3 void SetDigitalOutput (UINT32 *channel*, UINT32 *value*)

Set the value of a digital output. Set the value of a digital output to either one (true) or zero (false).

Parameters:

channel The channel being used for this digital output

value The 0/1 value set to the port.

6.12.2.4 void SetDigitalOutput (UINT32 *slot*, UINT32 *channel*, UINT32 *value*)

Set the value of a digital output. Set the value of a digital output to either one (true) or zero (false).

Parameters:

slot The slot this digital module is plugged into

channel The channel being used for this digital output

value The 0/1 value set to the port.

6.13 CDriverStation.cpp File Reference

```
#include "DriverStation.h"
#include "CDriverStation.h"
```

Functions

- float GetStickAxis (UINT32 stick, UINT32 axis)
- short GetStickButtons (UINT32 stick)
- float GetAnalogIn (UINT32 channel)
- bool GetDigitalIn (UINT32 channel)
- void SetDigitalOut (UINT32 channel, bool value)
- bool GetDigitalOut (UINT32 channel)
- bool IsDisabled ()
- bool IsAutonomous ()
- bool IsOperatorControl ()
- UINT32 GetPacketNumber ()
- UINT32 GetAlliance ()
- UINT32 GetLocation ()
- float GetBatteryVoltage ()

Variables

- static DriverStation * ds = NULL

6.13.1 Function Documentation

6.13.1.1 UINT32 GetAlliance ()

6.13.1.2 float GetAnalogIn (UINT32 *channel*)

Get an analog voltage from the Driver Station. The analog values are returned as UINT32 values for the Driver Station analog inputs. These inputs are typically used for advanced operator interfaces consisting of potentiometers or resistor networks representing values on a rotary switch.

Parameters:

channel The analog input channel on the driver station to read from. Valid range is 1 - 4.

Returns:

The analog voltage on the input.

6.13.1.3 float GetBatteryVoltage ()

Get the battery voltage on the robot

Returns:

the battery voltage in volts

6.13.1.4 bool GetDigitalIn (UINT32 *channel*)

Get values from the digital inputs on the Driver Station. Return digital values from the Drivers Station. These values are typically used for buttons and switches on advanced operator interfaces.

Parameters:

channel The digital input to get. Valid range is 1 - 8.

6.13.1.5 bool GetDigitalOut (UINT32 *channel*)

Get a value that was set for the digital outputs on the Driver Station.

Parameters:

channel The digital ouput to monitor. Valid range is 1 through 8.

Returns:

A digital value being output on the Drivers Station.

6.13.1.6 UINT32 GetLocation ()

6.13.1.7 UINT32 GetPacketNumber ()

Return the DS packet number. The packet number is the index of this set of data returned by the driver station. Each time new data is received, the packet number (included with the sent data) is returned.

6.13.1.8 float GetStickAxis (UINT32 *stick*, UINT32 *axis*)

Get the value of the axis on a joystick. This depends on the mapping of the joystick connected to the specified port.

Parameters:

stick The joystick to read.

axis The analog axis value to read from the joystick.

Returns:

The value of the axis on the joystick.

6.13.1.9 short GetStickButtons (UINT32 *stick*)

The state of the buttons on the joystick. 12 buttons (4 msb are unused) from the joystick.

Parameters:

stick The joystick to read.

Returns:

The state of the buttons on the joystick.

6.13.1.10 bool IsAutonomous ()

Returns flag for field state

Returns:

true if the field is in Autonomous mode

6.13.1.11 bool IsDisabled ()

Returns the robot state

Returns:

true if the robot is disabled

6.13.1.12 bool IsOperatorControl ()

Returns flag for field state

Returns:

true if the field is in Operator Control mode (teleop)

6.13.1.13 void SetDigitalOut (UINT32 *channel*, bool *value*)

Set a value for the digital outputs on the Driver Station.

Control digital outputs on the Drivers Station. These values are typically used for giving feedback on a custom operator station such as LEDs.

Parameters:

channel The digital output to set. Valid range is 1 - 8.

value The state to set the digital output.

6.13.2 Variable Documentation

6.13.2.1 DriverStation* ds = NULL [static]

6.14 CDriverStation.h File Reference

Functions

- float GetStickAxis (UINT32 stick, UINT32 axis)
- short GetStickButtons (UINT32 stick)
- float GetAnalogIn (UINT32 channel)
- bool GetDigitalIn (UINT32 channel)
- void SetDigitalOut (UINT32 channel, bool value)
- bool GetDigitalOut (UINT32 channel)
- bool IsDisabled ()
- bool IsAutonomous ()
- bool IsOperatorControl ()
- UINT32 GetPacketNumber ()
- UINT32 GetAlliance ()
- UINT32 GetLocation ()
- float GetBatteryVoltage ()

6.14.1 Function Documentation

6.14.1.1 UINT32 GetAlliance ()

6.14.1.2 float GetAnalogIn (UINT32 *channel*)

Get an analog voltage from the Driver Station. The analog values are returned as UINT32 values for the Driver Station analog inputs. These inputs are typically used for advanced operator interfaces consisting of potentiometers or resistor networks representing values on a rotary switch.

Parameters:

channel The analog input channel on the driver station to read from. Valid range is 1 - 4.

Returns:

The analog voltage on the input.

6.14.1.3 float GetBatteryVoltage ()

Get the battery voltage on the robot

Returns:

the battery voltage in volts

6.14.1.4 bool GetDigitalIn (UINT32 *channel*)

Get values from the digital inputs on the Driver Station. Return digital values from the Drivers Station. These values are typically used for buttons and switches on advanced operator interfaces.

Parameters:

channel The digital input to get. Valid range is 1 - 8.

6.14.1.5 bool GetDigitalOut (UINT32 *channel*)

Get a value that was set for the digital outputs on the Driver Station.

Parameters:

channel The digital ouput to monitor. Valid range is 1 through 8.

Returns:

A digital value being output on the Drivers Station.

6.14.1.6 UINT32 GetLocation ()

6.14.1.7 UINT32 GetPacketNumber ()

Return the DS packet number. The packet number is the index of this set of data returned by the driver station. Each time new data is received, the packet number (included with the sent data) is returned.

6.14.1.8 float GetStickAxis (UINT32 *stick*, UINT32 *axis*)

Get the value of the axis on a joystick. This depends on the mapping of the joystick connected to the specified port.

Parameters:

stick The joystick to read.

axis The analog axis value to read from the joystick.

Returns:

The value of the axis on the joystick.

6.14.1.9 short GetStickButtons (UINT32 *stick*)

The state of the buttons on the joystick. 12 buttons (4 msb are unused) from the joystick.

Parameters:

stick The joystick to read.

Returns:

The state of the buttons on the joystick.

6.14.1.10 bool IsAutonomous ()

Returns flag for field state

Returns:

true if the field is in Autonomous mode

6.14.1.11 bool IsDisabled ()

Returns the robot state

Returns:

true if the robot is disabled

6.14.1.12 bool IsOperatorControl ()

Returns flag for field state

Returns:

true if the field is in Operator Control mode (teleop)

6.14.1.13 void SetDigitalOut (UINT32 *channel*, bool *value*)

Set a value for the digital outputs on the Driver Station.

Control digital outputs on the Drivers Station. These values are typically used for giving feedback on a custom operator station such as LEDs.

Parameters:

channel The digital output to set. Valid range is 1 - 8.

value The state to set the digital output.

6.15 CEncoder.cpp File Reference

```
#include "Encoder.h"
#include "SensorBase.h"
#include "DigitalModule.h"
#include "CEncoder.h"
```

Functions

- static Encoder * AllocateEncoder (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- static Encoder * AllocateEncoder (UINT32 aChannel, UINT32 bChannel)
- void StartEncoder (UINT32 aChannel, UINT32 bChannel)
- void StartEncoder (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- INT32 GetEncoder (UINT32 aChannel, UINT32 bChannel)
- INT32 GetEncoder (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- void ResetEncoder (UINT32 aChannel, UINT32 bChannel)
- void ResetEncoder (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- void StopEncoder (UINT32 aChannel, UINT32 bChannel)
- void StopEncoder (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- double GetEncoderPeriod (UINT32 aChannel, UINT32 bChannel)
- double GetEncoderPeriod (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- void SetMaxEncoderPeriod (UINT32 aChannel, UINT32 bChannel, double maxPeriod)
- void SetMaxEncoderPeriod (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel, double maxPeriod)
- bool GetEncoderStopped (UINT32 aChannel, UINT32 bChannel)
- bool GetEncoderStopped (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- bool GetEncoderDirection (UINT32 aChannel, UINT32 bChannel)
- bool GetEncoderDirection (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- double GetEncoderDistance (UINT32 aChannel, UINT32 bChannel)
- double GetEncoderDistance (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- double GetEncoderRate (UINT32 aChannel, UINT32 bChannel)
- double GetEncoderRate (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- void SetMinEncoderRate (UINT32 aChannel, UINT32 bChannel, double minRate)
- void SetMinEncoderRate (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel, double minRate)
- void SetEncoderDistancePerPulse (UINT32 aChannel, UINT32 bChannel, double distancePerPulse)
- void SetEncoderDistancePerPulse (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel, double distancePerPulse)
- void SetEncoderReverseDirection (UINT32 aChannel, UINT32 bChannel, bool reverseDirection)
- void SetEncoderReverseDirection (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel, bool reverseDirection)
- void DeleteEncoder (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)
- void DeleteEncoder (UINT32 aChannel, UINT32 bChannel)

Variables

- static Encoder * encoders [SensorBase::kDigitalModules][SensorBase::kDigitalChannels]
- static bool initialized = false

6.15.1 Function Documentation

6.15.1.1 static Encoder* AllocateEncoder (UINT32 *aChannel*, UINT32 *bChannel*) [static]

Allocate the resources associated with this encoder. Allocate an Encoder object and cache the value in the associated table to find it in the future.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.2 static Encoder* AllocateEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*) [static]

Allocate the resources associated with this encoder. Allocate an Encoder object and cache the value in the associated table to find it in the future.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.3 void DeleteEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Free the resources associated with this encoder. Delete the Encoder object and the entries from the cache for this encoder.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.4 void DeleteEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Free the resources associated with this encoder. Delete the Encoder object and the entries from the cache for this encoder.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.5 INT32 GetEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Gets the current count. Returns the current count on the Encoder. This method compensates for the decoding type.

Deprecated

Use GetEncoderDistance() in favor of this method. This returns unscaled pulses and GetDistance() scales using value from SetEncoderDistancePerPulse().

Returns:

Current count from the Encoder.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.6 INT32 GetEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Gets the current count. Returns the current count on the Encoder. This method compensates for the decoding type.

Deprecated

Use GetEncoderDistance() in favor of this method. This returns unscaled pulses and GetDistance() scales using value from SetEncoderDistancePerPulse().

Returns:

Current count from the Encoder.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.7 bool GetEncoderDirection (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

The last direction the encoder value changed.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

Returns:

The last direction the encoder value changed.

6.15.1.8 bool GetEncoderDirection (UINT32 *aChannel*, UINT32 *bChannel*)

The last direction the encoder value changed.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

Returns:

The last direction the encoder value changed.

6.15.1.9 double GetEncoderDistance (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Get the distance the robot has driven since the last reset.

Returns:

The distance driven since the last reset as scaled by the value from SetEncoderDistancePerPulse().

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.10 double GetEncoderDistance (UINT32 *aChannel*, UINT32 *bChannel*)

Get the distance the robot has driven since the last reset.

Returns:

The distance driven since the last reset as scaled by the value from SetEncoderDistancePerPulse().

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.11 double GetEncoderPeriod (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Returns the period of the most recent pulse. Returns the period of the most recent Encoder pulse in seconds. This method compenstates for the decoding type.

Deprecated

Use GetEncoderRate() in favor of this method. This returns unscaled periods and GetEncoderRate() scales using value from SetEncoderDistancePerPulse().

Returns:

Period in seconds of the most recent pulse.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.12 double GetEncoderPeriod (UINT32 *aChannel*, UINT32 *bChannel*)

Returns the period of the most recent pulse. Returns the period of the most recent Encoder pulse in seconds. This method compenstates for the decoding type.

Deprecated

Use GetEncoderRate() in favor of this method. This returns unscaled periods and GetEncoderRate() scales using value from SetEncoderDistancePerPulse().

Returns:

Period in seconds of the most recent pulse.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.13 double GetEncoderRate (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Get the current rate of the encoder. Units are distance per second as scaled by the value from SetEncoderDistancePerPulse().

Returns:

The current rate of the encoder.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.14 double GetEncoderRate (UINT32 *aChannel*, UINT32 *bChannel*)

Get the current rate of the encoder. Units are distance per second as scaled by the value from SetEncoderDistancePerPulse().

Returns:

The current rate of the encoder.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.15 bool GetEncoderStopped (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Determine if the encoder is stopped. Using the MaxPeriod value, a boolean is returned that is true if the encoder is considered stopped and false if it is still moving. A stopped encoder is one where the most recent pulse width exceeds the MaxPeriod.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

Returns:

True if the encoder is considered stopped.

6.15.1.16 bool GetEncoderStopped (UINT32 *aChannel*, UINT32 *bChannel*)

Determine if the encoder is stopped. Using the MaxPeriod value, a boolean is returned that is true if the encoder is considered stopped and false if it is still moving. A stopped encoder is one where the most recent pulse width exceeds the MaxPeriod.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

Returns:

True if the encoder is considered stopped.

6.15.1.17 void ResetEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Reset the count for the encoder object. Resets the count to zero.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.18 void ResetEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Reset the count for the encoder object. Resets the count to zero.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.19 void SetEncoderDistancePerPulse (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*, double *distancePerPulse*)

Set the distance per pulse for this encoder. This sets the multiplier used to determine the distance driven based on the count value from the encoder. Do not include the decoding type in this scale. The library already compensates for the decoding type. Set this value based on the encoder's rated Pulses per Revolution and factor in gearing reductions following the encoder shaft. This distance can be in any units you like, linear or angular.

Parameters:

distancePerPulse The scale factor that will be used to convert pulses to useful units.

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.20 void SetEncoderDistancePerPulse (UINT32 *aChannel*, UINT32 *bChannel*, double *distancePerPulse*)

Set the distance per pulse for this encoder. This sets the multiplier used to determine the distance driven based on the count value from the encoder. Do not include the decoding type in this scale. The library already compensates for the decoding type. Set this value based on the encoder's rated Pulses per Revolution and factor in gearing reductions following the encoder shaft. This distance can be in any units you like, linear or angular.

Parameters:

distancePerPulse The scale factor that will be used to convert pulses to useful units.

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.21 void SetEncoderReverseDirection (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*, bool *reverseDirection*)

Set the direction sensing for this encoder. This sets the direction sensing on the encoder so that it couldl count in the correct software direction regardless of the mounting.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

reverseDirection true if the encoder direction should be reversed

6.15.1.22 void SetEncoderReverseDirection (UINT32 *aChannel*, UINT32 *bChannel*, bool *reverseDirection*)

Set the direction sensing for this encoder. This sets the direction sensing on the encoder so that it could count in the correct software direction regardless of the mounting.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

reverseDirection true if the encoder direction should be reversed

6.15.1.23 void SetMaxEncoderPeriod (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*, double *maxPeriod*)

Sets the maximum period for stopped detection. Sets the value that represents the maximum period of the Encoder before it will assume that the attached device is stopped. This timeout allows users to determine if the wheels or other shaft has stopped rotating. This method compensates for the decoding type.

Deprecated

Use SetEncoderMinRate() in favor of this method. This takes unscaled periods and SetMinEncoderRate() scales using value from SetEncoderDistancePerPulse().

Parameters:

maxPeriod The maximum time between rising and falling edges before the FPGA will report the device stopped. This is expressed in seconds.

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.24 void SetMaxEncoderPeriod (UINT32 *aChannel*, UINT32 *bChannel*, double *maxPeriod*)

Sets the maximum period for stopped detection. Sets the value that represents the maximum period of the Encoder before it will assume that the attached device is stopped. This timeout allows users to determine if the wheels or other shaft has stopped rotating. This method compensates for the decoding type.

Deprecated

Use SetEncoderMinRate() in favor of this method. This takes unscaled periods and SetMinEncoderRate() scales using value from SetEncoderDistancePerPulse().

Parameters:

maxPeriod The maximum time between rising and falling edges before the FPGA will report the device stopped. This is expressed in seconds.

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.25 void SetMinEncoderRate (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*, double *minRate*)

Set the minimum rate of the device before the hardware reports it stopped.

Parameters:

minRate The minimum rate. The units are in distance per second as scaled by the value from SetEncoderDistancePerPulse().

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.26 void SetMinEncoderRate (UINT32 *aChannel*, UINT32 *bChannel*, double *minRate*)

Set the minimum rate of the device before the hardware reports it stopped.

Parameters:

minRate The minimum rate. The units are in distance per second as scaled by the value from SetEncoderDistancePerPulse().

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.27 void StartEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Start the encoder counting.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.28 void StartEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Start the encoder counting.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.29 void StopEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Stops the counting for the encoder object. Stops the counting for the Encoder. It still retains the count, but it doesn't change with pulses until it is started again.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.1.30 void StopEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Stops the counting for the encoder object. Stops the counting for the Encoder. It still retains the count, but it doesn't change with pulses until it is started again.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.15.2 Variable Documentation

6.15.2.1 Encoder* encoders[SensorBase::kDigitalModules][SensorBase::kDigitalChannels] [static]

6.15.2.2 bool initialized = false [static]

6.16 CEncoder.h File Reference

Functions

- void StartEncoder (UINT32 aChannel, UINT32 bChannel)
- INT32 GetEncoder (UINT32 aChannel, UINT32 bChannel)
- void ResetEncoder (UINT32 aChannel, UINT32 bChannel)
- void StopEncoder (UINT32 aChannel, UINT32 bChannel)
- double GetEncoderPeriod (UINT32 aChannel, UINT32 bChannel)
- void SetMaxEncoderPeriod (UINT32 aChannel, UINT32 bChannel, double maxPeriod)
- bool GetEncoderStopped (UINT32 aChannel, UINT32 bChannel)
- bool GetEncoderDirection (UINT32 aChannel, UINT32 bChannel)
- double GetEncoderDistance (UINT32 aChannel, UINT32 bChannel)
- double GetEncoderRate (UINT32 aChannel, UINT32 bChannel)
- void SetMinEncoderRate (UINT32 aChannel, UINT32 bChannel, double minRate)
- void SetEncoderDistancePerPulse (UINT32 aChannel, UINT32 bChannel, double distancePerPulse)
- void SetEncoderReverseDirection (UINT32 aChannel, UINT32 bChannel, bool reversedDirection)
- void StartEncoder (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- INT32 GetEncoder (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- void ResetEncoder (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- void StopEncoder (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- double GetEncoderPeriod (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- void SetMaxEncoderPeriod (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel, double maxPeriod)
- bool GetEncoderStopped (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- bool GetEncoderDirection (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- double GetEncoderDistance (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- double GetEncoderRate (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel)
- void SetMinEncoderRate (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel, double minRate)
- void SetEncoderDistancePerPulse (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel, double distancePerPulse)
- void SetEncoderReverseDirection (UINT32 aslot, UINT32 aChannel, UINT32 bslot, UINT32 bChannel, bool reversedDirection)
- void DeleteEncoder (UINT32 aChannel, UINT32 bChannel)
- void DeleteEncoder (UINT32 aSlot, UINT32 aChannel, UINT32 bSlot, UINT32 bChannel)

6.16.1 Function Documentation

6.16.1.1 void DeleteEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Free the resources associated with this encoder. Delete the Encoder object and the entries from the cache for this encoder.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.2 void DeleteEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Free the resources associated with this encoder. Delete the Encoder object and the entries from the cache for this encoder.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.3 INT32 GetEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Gets the current count. Returns the current count on the Encoder. This method compensates for the decoding type.

Deprecated

Use GetEncoderDistance() in favor of this method. This returns unscaled pulses and GetDistance() scales using value from SetEncoderDistancePerPulse().

Returns:

Current count from the Encoder.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.4 INT32 GetEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Gets the current count. Returns the current count on the Encoder. This method compensates for the decoding type.

Deprecated

Use GetEncoderDistance() in favor of this method. This returns unscaled pulses and GetDistance() scales using value from SetEncoderDistancePerPulse().

Returns:

Current count from the Encoder.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.5 bool GetEncoderDirection (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

The last direction the encoder value changed.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

Returns:

The last direction the encoder value changed.

6.16.1.6 bool GetEncoderDirection (UINT32 *aChannel*, UINT32 *bChannel*)

The last direction the encoder value changed.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

Returns:

The last direction the encoder value changed.

6.16.1.7 double GetEncoderDistance (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Get the distance the robot has driven since the last reset.

Returns:

The distance driven since the last reset as scaled by the value from SetEncoderDistancePerPulse().

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.8 double GetEncoderDistance (UINT32 *aChannel*, UINT32 *bChannel*)

Get the distance the robot has driven since the last reset.

Returns:

The distance driven since the last reset as scaled by the value from SetEncoderDistancePerPulse().

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.9 double GetEncoderPeriod (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Returns the period of the most recent pulse. Returns the period of the most recent Encoder pulse in seconds. This method compenstates for the decoding type.

Deprecated

Use GetEncoderRate() in favor of this method. This returns unscaled periods and GetEncoderRate() scales using value from SetEncoderDistancePerPulse().

Returns:

Period in seconds of the most recent pulse.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.10 double GetEncoderPeriod (UINT32 *aChannel*, UINT32 *bChannel*)

Returns the period of the most recent pulse. Returns the period of the most recent Encoder pulse in seconds. This method compenstates for the decoding type.

Deprecated

Use GetEncoderRate() in favor of this method. This returns unscaled periods and GetEncoderRate() scales using value from SetEncoderDistancePerPulse().

Returns:

Period in seconds of the most recent pulse.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.11 double GetEncoderRate (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Get the current rate of the encoder. Units are distance per second as scaled by the value from SetEncoderDistancePerPulse().

Returns:

The current rate of the encoder.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.12 double GetEncoderRate (UINT32 *aChannel*, UINT32 *bChannel*)

Get the current rate of the encoder. Units are distance per second as scaled by the value from SetEncoderDistancePerPulse().

Returns:

The current rate of the encoder.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.13 bool GetEncoderStopped (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Determine if the encoder is stopped. Using the MaxPeriod value, a boolean is returned that is true if the encoder is considered stopped and false if it is still moving. A stopped encoder is one where the most recent pulse width exceeds the MaxPeriod.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

Returns:

True if the encoder is considered stopped.

6.16.1.14 bool GetEncoderStopped (UINT32 *aChannel*, UINT32 *bChannel*)

Determine if the encoder is stopped. Using the MaxPeriod value, a boolean is returned that is true if the encoder is considered stopped and false if it is still moving. A stopped encoder is one where the most recent pulse width exceeds the MaxPeriod.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

Returns:

True if the encoder is considered stopped.

6.16.1.15 void ResetEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Reset the count for the encoder object. Resets the count to zero.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.16 void ResetEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Reset the count for the encoder object. Resets the count to zero.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.17 void SetEncoderDistancePerPulse (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*, double *distancePerPulse*)

Set the distance per pulse for this encoder. This sets the multiplier used to determine the distance driven based on the count value from the encoder. Do not include the decoding type in this scale. The library already compensates for the decoding type. Set this value based on the encoder's rated Pulses per Revolution and factor in gearing reductions following the encoder shaft. This distance can be in any units you like, linear or angular.

Parameters:

distancePerPulse The scale factor that will be used to convert pulses to useful units.

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.18 void SetEncoderDistancePerPulse (UINT32 *aChannel*, UINT32 *bChannel*, double *distancePerPulse*)

Set the distance per pulse for this encoder. This sets the multiplier used to determine the distance driven based on the count value from the encoder. Do not include the decoding type in this scale. The library already compensates for the decoding type. Set this value based on the encoder's rated Pulses per Revolution and factor in gearing reductions following the encoder shaft. This distance can be in any units you like, linear or angular.

Parameters:

distancePerPulse The scale factor that will be used to convert pulses to useful units.

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.19 void SetEncoderReverseDirection (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*, bool *reverseDirection*)

Set the direction sensing for this encoder. This sets the direction sensing on the encoder so that it couldl count in the correct software direction regardless of the mounting.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

reverseDirection true if the encoder direction should be reversed

6.16.1.20 void SetEncoderReverseDirection (UINT32 *aChannel*, UINT32 *bChannel*, bool *reverseDirection*)

Set the direction sensing for this encoder. This sets the direction sensing on the encoder so that it could count in the correct software direction regardless of the mounting.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

reverseDirection true if the encoder direction should be reversed

6.16.1.21 void SetMaxEncoderPeriod (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*, double *maxPeriod*)

Sets the maximum period for stopped detection. Sets the value that represents the maximum period of the Encoder before it will assume that the attached device is stopped. This timeout allows users to determine if the wheels or other shaft has stopped rotating. This method compensates for the decoding type.

Deprecated

Use SetEncoderMinRate() in favor of this method. This takes unscaled periods and SetMinEncoderRate() scales using value from SetEncoderDistancePerPulse().

Parameters:

maxPeriod The maximum time between rising and falling edges before the FPGA will report the device stopped. This is expressed in seconds.

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.22 void SetMaxEncoderPeriod (UINT32 *aChannel*, UINT32 *bChannel*, double *maxPeriod*)

Sets the maximum period for stopped detection. Sets the value that represents the maximum period of the Encoder before it will assume that the attached device is stopped. This timeout allows users to determine if the wheels or other shaft has stopped rotating. This method compensates for the decoding type.

Deprecated

Use SetEncoderMinRate() in favor of this method. This takes unscaled periods and SetMinEncoderRate() scales using value from SetEncoderDistancePerPulse().

Parameters:

maxPeriod The maximum time between rising and falling edges before the FPGA will report the device stopped. This is expressed in seconds.

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.23 void SetMinEncoderRate (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*, double *minRate*)

Set the minimum rate of the device before the hardware reports it stopped.

Parameters:

minRate The minimum rate. The units are in distance per second as scaled by the value from SetEncoderDistancePerPulse().

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.24 void SetMinEncoderRate (UINT32 *aChannel*, UINT32 *bChannel*, double *minRate*)

Set the minimum rate of the device before the hardware reports it stopped.

Parameters:

minRate The minimum rate. The units are in distance per second as scaled by the value from SetEncoderDistancePerPulse().

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.25 void StartEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Start the encoder counting.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.26 void StartEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Start the encoder counting.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.27 void StopEncoder (UINT32 *aSlot*, UINT32 *aChannel*, UINT32 *bSlot*, UINT32 *bChannel*)

Stops the counting for the encoder object. Stops the counting for the Encoder. It still retains the count, but it doesn't change with pulses until it is started again.

Parameters:

aSlot The digital module slot for the A Channel on the encoder

aChannel The channel on the digital module for the A Channel of the encoder

bSlot The digital module slot for the B Channel on the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.16.1.28 void StopEncoder (UINT32 *aChannel*, UINT32 *bChannel*)

Stops the counting for the encoder object. Stops the counting for the Encoder. It still retains the count, but it doesn't change with pulses until it is started again.

Parameters:

aChannel The channel on the digital module for the A Channel of the encoder

bChannel The channel on the digital module for the B Channel of the encoder

6.17 CGearTooth.cpp File Reference

```
#include "CGearTooth.h"
#include "DigitalModule.h"
```

Functions

- static GearTooth * GTptr (UINT32 slot, UINT32 channel)
- void InitGearTooth (UINT32 slot, UINT32 channel, bool directionSensitive)
- void InitGearTooth (UINT32 channel, bool directionSensitive)
- void StartGearTooth (UINT32 slot, UINT32 channel)
- void StartGearTooth (UINT32 channel)
- void StopGearTooth (UINT32 slot, UINT32 channel)
- void StopGearTooth (UINT32 channel)
- INT32 GetGearTooth (UINT32 slot, UINT32 channel)
- INT32 GetGearTooth (UINT32 channel)
- void ResetGearTooth (UINT32 slot, UINT32 channel)
- void ResetGearTooth (UINT32 channel)
- void DeleteGearTooth (UINT32 slot, UINT32 channel)
- void DeleteGearTooth (UINT32 channel)

Variables

- static GearTooth * gearToothSensors [SensorBase::kChassisSlots][SensorBase::kDigitalChannels]
- static bool initialized = false

6.17.1 Function Documentation

6.17.1.1 void DeleteGearTooth (UINT32 *channel*)

Free the resources associated with this gear tooth sensor. Delete the underlying object and free the resources for this geartooth sensor.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.17.1.2 void DeleteGearTooth (UINT32 *slot*, UINT32 *channel*)

Free the resources associated with this gear tooth sensor. Delete the underlying object and free the resources for this geartooth sensor.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

6.17.1.3 INT32 GetGearTooth (UINT32 *channel*)

Get value from GearTooth sensor. Get the current count from the sensor.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.17.1.4 INT32 GetGearTooth (UINT32 *slot*, UINT32 *channel*)

Get value from GearTooth sensor. Get the current count from the sensor.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

6.17.1.5 static GearTooth* GTptr (UINT32 *slot*, UINT32 *channel*) [static]

Get a pointer to the gear tooth sensor given a slot and a channel. This is an internal routine to allocate (if necessary) a gear tooth object from inputs.

Parameters:

slot The slot the GearTooth sensor is plugged into.

channel The channel the GearTooth sensor is plugged into.

6.17.1.6 void InitGearTooth (UINT32 *channel*, bool *directionSensitive*)

Initialize the gear tooth sensor.

Parameters:

channel The digital I/O channel the sensor is plugged into

directionSensitive True if this geartooth sensor can differentiate between foward and backward movement.

6.17.1.7 void InitGearTooth (UINT32 *slot*, UINT32 *channel*, bool *directionSensitive*)

Initialize the gear tooth sensor.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

directionSensitive True if this geartooth sensor can differentiate between foward and backward movement.

6.17.1.8 void ResetGearTooth (UINT32 *channel*)

Reset the GearTooth sensor. Reset the count to zero for the gear tooth sensor.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.17.1.9 void ResetGearTooth (UINT32 *slot*, UINT32 *channel*)

Reset the GearTooth sensor. Reset the count to zero for the gear tooth sensor.

Parameters:

slot The slot the digital module is plugged into
channel The digital I/O channel the sensor is plugged into

6.17.1.10 void StartGearTooth (UINT32 *channel*)

Start the GearTooth sensor counting. Start the counting for the geartooth sensor. Before this, the sensor is allocated but not counting pulses.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.17.1.11 void StartGearTooth (UINT32 *slot*, UINT32 *channel*)

Start the GearTooth sensor counting. Start the counting for the geartooth sensor. Before this, the sensor is allocated but not counting pulses.

Parameters:

slot The slot the digital module is plugged into
channel The digital I/O channel the sensor is plugged into

6.17.1.12 void StopGearTooth (UINT32 *channel*)

Stop the gear tooth sensor from counting. The counting is disabled on the underlying Counter object.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.17.1.13 void StopGearTooth (UINT32 *slot*, UINT32 *channel*)

Stop the gear tooth sensor from counting. The counting is disabled on the underlying Counter object.

Parameters:

slot The slot the digital module is plugged into
channel The digital I/O channel the sensor is plugged into

6.17.2 Variable Documentation

6.17.2.1 GearTooth* gearToothSensors[SensorBase::kChassisSlots][SensorBase::kDigitalChannels] [static]

6.17.2.2 bool initialized = false [static]

6.18 CGearTooth.h File Reference

```
#include "GearTooth.h"
```

Functions

- void InitGearTooth (UINT32 channel, bool directionSensitive)
- void InitGearTooth (UINT32 slot, UINT32 channel, bool directionSensitive)
- void StartGearTooth (UINT32 channel)
- void StartGearTooth (UINT32 slot, UINT32 channel)
- void StopGearTooth (UINT32 channel)
- void StopGearTooth (UINT32 slot, UINT32 channel)
- INT32 GetGearTooth (UINT32 channel)
- INT32 GetGearTooth (UINT32 slot, UINT32 channel)
- void ResetGearTooth (UINT32 channel)
- void ResetGearTooth (UINT32 slot, UINT32 channel)
- void DeleteGearTooth (UINT32 channel)
- void DeleteGearTooth (UINT32 slot, UINT32 channel)

6.18.1 Function Documentation

6.18.1.1 void DeleteGearTooth (UINT32 *slot*, UINT32 *channel*)

Free the resources associated with this gear tooth sensor. Delete the underlying object and free the resources for this geartooth sensor.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

6.18.1.2 void DeleteGearTooth (UINT32 *channel*)

Free the resources associated with this gear tooth sensor. Delete the underlying object and free the resources for this geartooth sensor.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.18.1.3 INT32 GetGearTooth (UINT32 *slot*, UINT32 *channel*)

Get value from GearTooth sensor. Get the current count from the sensor.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

6.18.1.4 INT32 GetGearTooth (UINT32 *channel*)

Get value from GearTooth sensor. Get the current count from the sensor.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.18.1.5 void InitGearTooth (UINT32 *slot*, UINT32 *channel*, bool *directionSensitive*)

Initialize the gear tooth sensor.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

directionSensitive True if this geartooth sensor can differentiate between foward and backward movement.

6.18.1.6 void InitGearTooth (UINT32 *channel*, bool *directionSensitive*)

Initialize the gear tooth sensor.

Parameters:

channel The digital I/O channel the sensor is plugged into

directionSensitive True if this geartooth sensor can differentiate between foward and backward movement.

6.18.1.7 void ResetGearTooth (UINT32 *slot*, UINT32 *channel*)

Reset the GearTooth sensor. Reset the count to zero for the gear tooth sensor.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

6.18.1.8 void ResetGearTooth (UINT32 *channel*)

Reset the GearTooth sensor. Reset the count to zero for the gear tooth sensor.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.18.1.9 void StartGearTooth (UINT32 *slot*, UINT32 *channel*)

Start the GearTooth sensor counting. Start the counting for the geartooth sensor. Before this, the sensor is allocated but not counting pulses.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

6.18.1.10 void StartGearTooth (UINT32 *channel*)

Start the GearTooth sensor counting. Start the counting for the geartooth sensor. Before this, the sensor is allocated but not counting pulses.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.18.1.11 void StopGearTooth (UINT32 *slot*, UINT32 *channel*)

Stop the gear tooth sensor from counting. The counting is disabled on the underlying Counter object.

Parameters:

slot The slot the digital module is plugged into

channel The digital I/O channel the sensor is plugged into

6.18.1.12 void StopGearTooth (UINT32 *channel*)

Stop the gear tooth sensor from counting. The counting is disabled on the underlying Counter object.

Parameters:

channel The digital I/O channel the sensor is plugged into

6.19 CGyro.cpp File Reference

```
#include "CGyro.h"
#include "Gyro.h"
```

Functions

- static Gyro * AllocateGyro (UINT32 slot, UINT32 channel)
- void InitGyro (UINT32 slot, UINT32 channel)
- void InitGyro (UINT32 channel)
- float GetGyroAngle (UINT32 slot, UINT32 channel)
- float GetGyroAngle (UINT32 channel)
- void ResetGyro (UINT32 slot, UINT32 channel)
- void ResetGyro (UINT32 channel)
- void SetGyroSensitivity (UINT32 slot, UINT32 channel, float voltsPerDegreePerSecond)
- void SetGyroSensitivity (UINT32 channel, float voltsPerDegreePerSecond)
- void DeleteGyro (UINT32 slot, UINT32 channel)
- void DeleteGyro (UINT32 channel)

Variables

- static Gyro * gyros [2] = {NULL, NULL}

6.19.1 Function Documentation

6.19.1.1 static Gyro* AllocateGyro (UINT32 *slot*, UINT32 *channel*) [static]

Allocate resoures for a Gyro.

This is an internal routine and not used outside of this module.

Parameters:

slot The analog module that the gyro is connected to. Must be slot 1 on the current hardware implementation.

channel The analog channel the gyro is connected to. Must be channel 1 or 2 only (the only ones with the attached accumulator)

6.19.1.2 void DeleteGyro (UINT32 *channel*)

6.19.1.3 void DeleteGyro (UINT32 *slot*, UINT32 *channel*)

Free the resources associated with this Gyro Free the Gyro object and the reservation for this slot/channel.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

6.19.1.4 float GetGyroAngle (UINT32 *channel*)

Return the actual angle in degrees that the robot is currently facing.

The angle is based on the current accumulator value corrected by the oversampling rate, the gyro type and the A/D calibration values. The angle is continuous, that is can go beyond 360 degrees. This make algorithms that wouldn't want to see a discontinuity in the gyro output as it sweeps past 0 on the second time around.

Parameters:

channel The analog channel the gyro is plugged into

Returns:

the current heading of the robot in degrees. This heading is based on integration of the returned rate from the gyro.

6.19.1.5 float GetGyroAngle (UINT32 *slot*, UINT32 *channel*)

Return the actual angle in degrees that the robot is currently facing.

The angle is based on the current accumulator value corrected by the oversampling rate, the gyro type and the A/D calibration values. The angle is continuous, that is can go beyond 360 degrees. This make algorithms that wouldn't want to see a discontinuity in the gyro output as it sweeps past 0 on the second time around.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

Returns:

the current heading of the robot in degrees. This heading is based on integration of the returned rate from the gyro.

6.19.1.6 void InitGyro (UINT32 *channel*)

Initialize the gyro. Calibrate the gyro by running for a number of samples and computing the center value for this part. Then use the center value as the Accumulator center value for subsequent measurements. It's important to make sure that the robot is not moving while the centering calculations are in progress, this is typically done when the robot is first turned on while it's sitting at rest before the competition starts.

Parameters:

channel The analog channel the gyro is plugged into

6.19.1.7 void InitGyro (UINT32 *slot*, UINT32 *channel*)

Initialize the gyro. Calibrate the gyro by running for a number of samples and computing the center value for this part. Then use the center value as the Accumulator center value for subsequent measurements. It's important to make sure that the robot is not moving while the centering calculations are in progress, this is typically done when the robot is first turned on while it's sitting at rest before the competition starts.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

6.19.1.8 void ResetGyro (UINT32 *channel*)

Reset the gyro. Resets the gyro to a heading of zero. This can be used if there is significant drift in the gyro and it needs to be recalibrated after it has been running.

Parameters:

channel The analog channel the gyro is plugged into

6.19.1.9 void ResetGyro (UINT32 *slot*, UINT32 *channel*)

Reset the gyro. Resets the gyro to a heading of zero. This can be used if there is significant drift in the gyro and it needs to be recalibrated after it has been running.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

6.19.1.10 void SetGyroSensitivity (UINT32 *channel*, float *voltsPerDegreePerSecond*)

Set the gyro type based on the sensitivity. This takes the number of volts/degree/second sensitivity of the gyro and uses it in subsequent calculations to allow the code to work with multiple gyros.

Parameters:

channel The analog channel the gyro is plugged into

voltsPerDegreePerSecond The type of gyro specified as the voltage that represents one degree/second.

6.19.1.11 void SetGyroSensitivity (UINT32 *slot*, UINT32 *channel*, float *voltsPerDegreePerSecond*)

Set the gyro type based on the sensitivity. This takes the number of volts/degree/second sensitivity of the gyro and uses it in subsequent calculations to allow the code to work with multiple gyros.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

voltsPerDegreePerSecond The type of gyro specified as the voltage that represents one degree/second.

6.19.2 Variable Documentation

6.19.2.1 Gyro* gyros[2] = {NULL, NULL} [static]

6.20 CGyro.h File Reference

```
#include <VxWorks.h>
```

Functions

- void InitGyro (UINT32 slot, UINT32 channel)
- void InitGyro (UINT32 channel)
- float GetGyroAngle (UINT32 channel)
- float GetGyroAngle (UINT32 slot, UINT32 channel)
- void ResetGyro (UINT32 channel)
- void ResetGyro (UINT32 slot, UINT32 channel)
- void SetGyroSensitivity (UINT32 slot, UINT32 channel, float voltsPerDegreePerSecond)
- void SetGyroSensitivity (UINT32 channel, float voltsPerDegreePerSecond)
- void DeleteGyro (UINT32 slot, UINT32 channel)
- void DeleteGyro (UINT32 channel)

6.20.1 Function Documentation

6.20.1.1 void DeleteGyro (UINT32 *channel*)

6.20.1.2 void DeleteGyro (UINT32 *slot*, UINT32 *channel*)

Free the resources associated with this Gyro Free the Gyro object and the reservation for this slot/channel.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

6.20.1.3 float GetGyroAngle (UINT32 *slot*, UINT32 *channel*)

Return the actual angle in degrees that the robot is currently facing.

The angle is based on the current accumulator value corrected by the oversampling rate, the gyro type and the A/D calibration values. The angle is continuous, that is can go beyond 360 degrees. This make algorithms that wouldn't want to see a discontinuity in the gyro output as it sweeps past 0 on the second time around.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

Returns:

the current heading of the robot in degrees. This heading is based on integration of the returned rate from the gyro.

6.20.1.4 float GetGyroAngle (UINT32 *channel*)

Return the actual angle in degrees that the robot is currently facing.

The angle is based on the current accumulator value corrected by the oversampling rate, the gyro type and the A/D calibration values. The angle is continuous, that is can go beyond 360 degrees. This make algorithms that wouldn't want to see a discontinuity in the gyro output as it sweeps past 0 on the second time around.

Parameters:

channel The analog channel the gyro is plugged into

Returns:

the current heading of the robot in degrees. This heading is based on integration of the returned rate from the gyro.

6.20.1.5 void InitGyro (UINT32 *channel*)

Initialize the gyro. Calibrate the gyro by running for a number of samples and computing the center value for this part. Then use the center value as the Accumulator center value for subsequent measurements. It's important to make sure that the robot is not moving while the centering calculations are in progress, this is typically done when the robot is first turned on while it's sitting at rest before the competition starts.

Parameters:

channel The analog channel the gyro is plugged into

6.20.1.6 void InitGyro (UINT32 *slot*, UINT32 *channel*)

Initialize the gyro. Calibrate the gyro by running for a number of samples and computing the center value for this part. Then use the center value as the Accumulator center value for subsequent measurements. It's important to make sure that the robot is not moving while the centering calculations are in progress, this is typically done when the robot is first turned on while it's sitting at rest before the competition starts.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

6.20.1.7 void ResetGyro (UINT32 *slot*, UINT32 *channel*)

Reset the gyro. Resets the gyro to a heading of zero. This can be used if there is significant drift in the gyro and it needs to be recalibrated after it has been running.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

6.20.1.8 void ResetGyro (UINT32 *channel*)

Reset the gyro. Resets the gyro to a heading of zero. This can be used if there is significant drift in the gyro and it needs to be recalibrated after it has been running.

Parameters:

channel The analog channel the gyro is plugged into

6.20.1.9 void SetGyroSensitivity (UINT32 *channel*, float *voltsPerDegreePerSecond*)

Set the gyro type based on the sensitivity. This takes the number of volts/degree/second sensitivity of the gyro and uses it in subsequent calculations to allow the code to work with multiple gyros.

Parameters:

channel The analog channel the gyro is plugged into

voltsPerDegreePerSecond The type of gyro specified as the voltage that represents one degree/second.

6.20.1.10 void SetGyroSensitivity (UINT32 *slot*, UINT32 *channel*, float *voltsPerDegreePerSecond*)

Set the gyro type based on the sensitivity. This takes the number of volts/degree/second sensitivity of the gyro and uses it in subsequent calculations to allow the code to work with multiple gyros.

Parameters:

slot The slot the analog module is connected to

channel The analog channel the gyro is plugged into

voltsPerDegreePerSecond The type of gyro specified as the voltage that represents one degree/second.

6.21 CJaguar.cpp File Reference

```
#include "../WPILib.h"
#include "CJaguar.h"
#include "CWrappers.h"
#include "CPWM.h"
```

Functions

- static SensorBase * CreateJaguar (UINT32 slot, UINT32 channel)
- void SetJaguarSpeed (UINT32 slot, UINT32 channel, float speed)
- void SetJaguarSpeed (UINT32 channel, float speed)
- void SetJaguarRaw (UINT32 channel, UINT8 value)
- UINT8 GetJaguarRaw (UINT32 channel)
- void SetJaguarRaw (UINT32 slot, UINT32 channel, UINT8 value)
- UINT8 GetJaguarRaw (UINT32 slot, UINT32 channel)
- void DeleteJaguar (UINT32 slot, UINT32 channel)
- void DeleteJaguar (UINT32 channel)

6.21.1 Function Documentation

6.21.1.1 static SensorBase* CreateJaguar (UINT32 *slot*, UINT32 *channel*) [static]

Create a Jaguar speed controller object. Allocate the object itself. This is a callback from the CPWM.cpp code to create the actual specific PWM object type.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

6.21.1.2 void DeleteJaguar (UINT32 *channel*)

Free the underlying Jaguar object. Free the underlying object and free the associated resources.

Parameters:

channel The PWM channel connected to this speed controller

6.21.1.3 void DeleteJaguar (UINT32 *slot*, UINT32 *channel*)

Free the underlying Jaguar object. Free the underlying object and free the associated resources.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

6.21.1.4 UINT8 GetJaguarRaw (UINT32 *slot*, UINT32 *channel*)

Get the PWM value directly from the hardware.

Read a raw value from a PWM channel.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

Returns:

Raw PWM control value. Range: 0 - 255.

6.21.1.5 UINT8 GetJaguarRaw (UINT32 *channel*)

Get the PWM value directly from the hardware.

Read a raw value from a PWM channel.

Parameters:

channel The PWM channel connected to this speed controller

Returns:

Raw PWM control value. Range: 0 - 255.

6.21.1.6 void SetJaguarRaw (UINT32 *slot*, UINT32 *channel*, UINT8 *value*)

Set the PWM value directly to the hardware.

Write a raw value to a PWM channel.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

value Raw PWM value. Range 0 - 255.

6.21.1.7 void SetJaguarRaw (UINT32 *channel*, UINT8 *value*)

Set the PWM value directly to the hardware.

Write a raw value to a PWM channel.

Parameters:

channel The PWM channel connected to this speed controller

value Raw PWM value. Range 0 - 255.

6.21.1.8 void SetJaguarSpeed (UINT32 *channel*, float *speed*)

Set the PWM value.

The PWM value is set using a range of -1.0 to 1.0, appropriately scaling the value for the FPGA.

Parameters:

channel The PWM channel connected to this speed controller

speed The speed value between -1.0 and 1.0 to set.

6.21.1.9 void SetJaguarSpeed (UINT32 *slot*, UINT32 *channel*, float *speed*)

Set the PWM value.

The PWM value is set using a range of -1.0 to 1.0, appropriately scaling the value for the FPGA.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

speed The speed value between -1.0 and 1.0 to set.

6.22 CJaguar.h File Reference

Functions

- void SetJaguarSpeed (UINT32 module, UINT32 channel, float speed)
- void SetJaguarSpeed (UINT32 channel, float speed)
- void SetJaguarRaw (UINT32 channel, UINT8 value)
- UINT8 GetJaguarRaw (UINT32 channel)
- void SetJaguarRaw (UINT32 module, UINT32 channel, UINT8 value)
- UINT8 GetJaguarRaw (UINT32 module, UINT32 channel)
- void DeleteJaguar (UINT32 module, UINT32 channel)
- void DeleteJaguar (UINT32 channel)

6.22.1 Function Documentation

6.22.1.1 void DeleteJaguar (UINT32 *channel*)

Free the underlying Jaguar object. Free the underlying object and free the associated resources.

Parameters:

channel The PWM channel connected to this speed controller

6.22.1.2 void DeleteJaguar (UINT32 *slot*, UINT32 *channel*)

Free the underlying Jaguar object. Free the underlying object and free the associated resources.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

6.22.1.3 UINT8 GetJaguarRaw (UINT32 *slot*, UINT32 *channel*)

Get the PWM value directly from the hardware.

Read a raw value from a PWM channel.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

Returns:

Raw PWM control value. Range: 0 - 255.

6.22.1.4 UINT8 GetJaguarRaw (UINT32 *channel*)

Get the PWM value directly from the hardware.

Read a raw value from a PWM channel.

Parameters:

channel The PWM channel connected to this speed controller

Returns:

Raw PWM control value. Range: 0 - 255.

6.22.1.5 void SetJaguarRaw (UINT32 *slot*, UINT32 *channel*, UINT8 *value*)

Set the PWM value directly to the hardware.

Write a raw value to a PWM channel.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

value Raw PWM value. Range 0 - 255.

6.22.1.6 void SetJaguarRaw (UINT32 *channel*, UINT8 *value*)

Set the PWM value directly to the hardware.

Write a raw value to a PWM channel.

Parameters:

channel The PWM channel connected to this speed controller

value Raw PWM value. Range 0 - 255.

6.22.1.7 void SetJaguarSpeed (UINT32 *channel*, float *speed*)

Set the PWM value.

The PWM value is set using a range of -1.0 to 1.0, appropriately scaling the value for the FPGA.

Parameters:

channel The PWM channel connected to this speed controller

speed The speed value between -1.0 and 1.0 to set.

6.22.1.8 void SetJaguarSpeed (UINT32 *slot*, UINT32 *channel*, float *speed*)

Set the PWM value.

The PWM value is set using a range of -1.0 to 1.0, appropriately scaling the value for the FPGA.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel connected to this speed controller

speed The speed value between -1.0 and 1.0 to set.

6.23 CJoystick.cpp File Reference

```
#include "Joystick.h"
#include "CJoystick.h"
```

Functions

- static Joystick * getJoystick (UINT32 port)
- UINT32 GetAxisChannel (UINT32 port, AxisType axis)
- void SetAxisChannel (UINT32 port, AxisType axis, UINT32 channel)
- float GetX (UINT32 port, JoystickHand hand)
- float GetY (UINT32 port, JoystickHand hand)
- float GetZ (UINT32 port)
- float GetTwist (UINT32 port)
- float GetThrottle (UINT32 port)
- float GetAxis (UINT32 port, AxisType axis)
- float GetRawAxis (UINT32 port, UINT32 axis)
- bool GetTrigger (UINT32 port, JoystickHand hand)
- bool GetTop (UINT32 port, JoystickHand hand)
- bool GetBumper (UINT32 port, JoystickHand hand)
- bool GetButton (UINT32 port, ButtonType button)
- bool GetRawButton (UINT32 port, UINT32 button)

Variables

- static Joystick * joysticks [4]
- static bool initialized = false

6.23.1 Function Documentation

6.23.1.1 float GetAxis (UINT32 *port*, AxisType *axis*)

For the current joystick, return the axis determined by the argument.

This is for cases where the joystick axis is returned programatically, otherwise one of the previous functions would be preferable (for example GetX()).

Parameters:

port The USB port for this joystick.

axis The axis to read.

Returns:

The value of the axis.

6.23.1.2 UINT32 GetAxisChannel (UINT32 *port*, AxisType *axis*)

Get the channel currently associated with the specified axis.

Parameters:

port The USB port for this joystick.

axis The axis to look up the channel for.

Returns:

The channel fr the axis.

6.23.1.3 bool GetBumper (UINT32 *port*, JoystickHand *hand*)

This is not supported for the Joystick. This method is only here to complete the GenericHID interface.

Parameters:

port The USB port for this joystick.

6.23.1.4 bool GetButton (UINT32 *port*, ButtonType *button*)

Get buttons based on an enumerated type.

The button type will be looked up in the list of buttons and then read.

Parameters:

port The USB port for this joystick.

button The type of button to read.

Returns:

The state of the button.

6.23.1.5 static Joystick* getJoystick (UINT32 *port*) [static]

Get the joystick associated with a port. An internal function that will return the joystick object associated with a given joystick port number. On the first call, all four joysticks are preallocated.

Parameters:

port The joystick (USB) port number

6.23.1.6 float GetRawAxis (UINT32 *port*, UINT32 *axis*)

Get the value of the axis.

Parameters:

port The USB port for this joystick.

axis The axis to read [1-6].

Returns:

The value of the axis.

6.23.1.7 bool GetRawButton (UINT32 *port*, UINT32 *button*)

Get the button value for buttons 1 through 12.

The buttons are returned in a single 16 bit value with one bit representing the state of each button. The appropriate button is returned as a boolean value.

Parameters:

port The USB port for this joystick.

button The button number to be read.

Returns:

The state of the button.

6.23.1.8 float GetThrottle (UINT32 *port*)

Get the throttle value of the current joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.23.1.9 bool GetTop (UINT32 *port*, JoystickHand *hand*)

Read the state of the top button on the joystick.

Look up which button has been assigned to the top and read its state.

Parameters:

port The USB port for this joystick.

hand This parameter is ignored for the Joystick class and is only here to complete the GenericHID interface.

Returns:

The state of the top button.

6.23.1.10 bool GetTrigger (UINT32 *port*, JoystickHand *hand*)

Read the state of the trigger on the joystick.

Look up which button has been assigned to the trigger and read its state.

Parameters:

port The USB port for this joystick.

hand This parameter is ignored for the Joystick class and is only here to complete the GenericHID interface.

Returns:

The state of the trigger.

6.23.1.11 float GetTwist (UINT32 *port*)

Get the twist value of the current joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.23.1.12 float GetX (UINT32 *port*, JoystickHand *hand*)

Get the X value of the joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.23.1.13 float GetY (UINT32 *port*, JoystickHand *hand*)

Get the Y value of the joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.23.1.14 float GetZ (UINT32 *port*)

Get the Z value of the current joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.23.1.15 void SetAxisChannel (UINT32 *port*, AxisType *axis*, UINT32 *channel*)

Set the channel associated with a specified axis.

Parameters:

port The USB port for this joystick.

axis The axis to set the channel for.

channel The channel to set the axis to.

6.23.2 Variable Documentation

6.23.2.1 bool initialized = false [static]

6.23.2.2 Joystick* joysticks[4] [static]

6.24 CJoystick.h File Reference

Enumerations

- enum JoystickHand { kLeftHand = 0, kRightHand = 1 }
- enum AxisType {
  kXAxis, kYAxis, kZAxis, kTwistAxis,
  kThrottleAxis, kNumAxisTypes }
- enum ButtonType { kTriggerButton, kTopButton, kNumButtonTypes }

Functions

- UINT32 GetAxisChannel (UINT32 port, AxisType axis)
- void SetAxisChannel (UINT32 port, AxisType axis, UINT32 channel)
- float GetX (UINT32 port, JoystickHand hand=kRightHand)
- float GetY (UINT32 port, JoystickHand hand=kRightHand)
- float GetZ (UINT32 port)
- float GetTwist (UINT32 port)
- float GetThrottle (UINT32 port)
- float GetAxis (UINT32 port, AxisType axis)
- float GetRawAxis (UINT32 port, UINT32 axis)
- bool GetTrigger (UINT32 port, JoystickHand hand=kRightHand)
- bool GetTop (UINT32 port, JoystickHand hand=kRightHand)
- bool GetBumper (UINT32 port, JoystickHand hand=kRightHand)
- bool GetButton (UINT32 port, ButtonType button)
- bool GetRawButton (UINT32 port, UINT32 button)

Variables

- static const UINT32 kDefaultXAxis = 1
- static const UINT32 kDefaultYAxis = 2
- static const UINT32 kDefaultZAxis = 3
- static const UINT32 kDefaultTwistAxis = 4
- static const UINT32 kDefaultThrottleAxis = 3
- static const UINT32 kDefaultTriggerButton = 1
- static const UINT32 kDefaultTopButton = 2

6.24.1 Enumeration Type Documentation

6.24.1.1 enum AxisType

Enumerator:

kXAxis

kYAxis

kZAxis

kTwistAxis

kThrottleAxis

kNumAxisTypes

6.24.1.2 enum ButtonType

Enumerator:

kTriggerButton
kTopButton
kNumButtonTypes

6.24.1.3 enum JoystickHand

Enumerator:

kLeftHand
kRightHand

6.24.2 Function Documentation

6.24.2.1 float GetAxis (UINT32 *port*, AxisType *axis*)

For the current joystick, return the axis determined by the argument.

This is for cases where the joystick axis is returned programatically, otherwise one of the previous functions would be preferable (for example GetX()).

Parameters:

port The USB port for this joystick.
axis The axis to read.

Returns:

The value of the axis.

6.24.2.2 UINT32 GetAxisChannel (UINT32 *port*, AxisType *axis*)

Get the channel currently associated with the specified axis.

Parameters:

port The USB port for this joystick.
axis The axis to look up the channel for.

Returns:

The channel fr the axis.

6.24.2.3 bool GetBumper (UINT32 *port*, JoystickHand *hand*)

This is not supported for the Joystick. This method is only here to complete the GenericHID interface.

Parameters:

port The USB port for this joystick.

6.24.2.4 bool GetButton (UINT32 *port*, ButtonType *button*)

Get buttons based on an enumerated type.

The button type will be looked up in the list of buttons and then read.

Parameters:

port The USB port for this joystick.

button The type of button to read.

Returns:

The state of the button.

6.24.2.5 float GetRawAxis (UINT32 *port*, UINT32 *axis*)

Get the value of the axis.

Parameters:

port The USB port for this joystick.

axis The axis to read [1-6].

Returns:

The value of the axis.

6.24.2.6 bool GetRawButton (UINT32 *port*, UINT32 *button*)

Get the button value for buttons 1 through 12.

The buttons are returned in a single 16 bit value with one bit representing the state of each button. The appropriate button is returned as a boolean value.

Parameters:

port The USB port for this joystick.

button The button number to be read.

Returns:

The state of the button.

6.24.2.7 float GetThrottle (UINT32 *port*)

Get the throttle value of the current joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.24.2.8 bool GetTop (UINT32 *port*, JoystickHand *hand*)

Read the state of the top button on the joystick.

Look up which button has been assigned to the top and read its state.

Parameters:

port The USB port for this joystick.

hand This parameter is ignored for the Joystick class and is only here to complete the GenericHID interface.

Returns:

The state of the top button.

6.24.2.9 bool GetTrigger (UINT32 *port*, JoystickHand *hand*)

Read the state of the trigger on the joystick.

Look up which button has been assigned to the trigger and read its state.

Parameters:

port The USB port for this joystick.

hand This parameter is ignored for the Joystick class and is only here to complete the GenericHID interface.

Returns:

The state of the trigger.

6.24.2.10 float GetTwist (UINT32 *port*)

Get the twist value of the current joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.24.2.11 float GetX (UINT32 *port*, JoystickHand *hand*)

Get the X value of the joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.24.2.12 float GetY (UINT32 *port*, JoystickHand *hand*)

Get the Y value of the joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.24.2.13 float GetZ (UINT32 *port*)

Get the Z value of the current joystick. This depends on the mapping of the joystick connected to the current port.

Parameters:

port The USB port for this joystick.

6.24.2.14 void SetAxisChannel (UINT32 *port*, AxisType *axis*, UINT32 *channel*)

Set the channel associated with a specified axis.

Parameters:

port The USB port for this joystick.

axis The axis to set the channel for.

channel The channel to set the axis to.

6.24.3 Variable Documentation

6.24.3.1 const UINT32 kDefaultThrottleAxis = 3 [static]

6.24.3.2 const UINT32 kDefaultTopButton = 2 [static]

6.24.3.3 const UINT32 kDefaultTriggerButton = 1 [static]

6.24.3.4 const UINT32 kDefaultTwistAxis = 4 [static]

6.24.3.5 const UINT32 kDefaultXAxis = 1 [static]

6.24.3.6 const UINT32 kDefaultYAxis = 2 [static]

6.24.3.7 const UINT32 kDefaultZAxis = 3 [static]

6.25 CPWM.cpp File Reference

```
#include "CPWM.h"
#include "../PWM.h"
#include "CWrappers.h"
#include "../DigitalModule.h"
```

Functions

- PWM * AllocatePWM (UINT32 module, UINT32 channel, SensorCreator createObject)
- PWM * AllocatePWM (UINT32 channel, SensorCreator createObject)
- void DeletePWM (UINT32 slot, UINT32 channel)
- void DeletePWM (UINT32 channel)

Variables

- static bool PWMsInitialized = false
- static PWM * PWMs [SensorBase::kDigitalModules][SensorBase::kPwmChannels]

6.25.1 Function Documentation

6.25.1.1 PWM* AllocatePWM (UINT32 *channel*, SensorCreator *createObject*)

Alloate a PWM based object

Allocate an instance of a PWM based object. This code is shared between the subclasses of PWM and is not usually created as a standalone object.

Parameters:

channel The PWM channel for this PWM object

createObject The function callback in the subclass object that actually creates an instance of the appropriate class.

6.25.1.2 PWM* AllocatePWM (UINT32 *module*, UINT32 *channel*, SensorCreator *createObject*)

Alloate a PWM based object

Allocate an instance of a PWM based object. This code is shared between the subclasses of PWM and is not usually created as a standalone object.

Parameters:

module The slot the digital module is plugged into that corresponds to this serial port

channel The PWM channel for this PWM object

createObject The function callback in the subclass object that actually creates an instance of the appropriate class.

6.25.1.3 void DeletePWM (UINT32 *channel*)

Delete a PWM Delete a PWM and free up all the associated resources for this object.

Parameters:

channel The PWM channel for this PWM object

6.25.1.4 void DeletePWM (UINT32 *slot*, UINT32 *channel*)

Delete a PWM Delete a PWM and free up all the associated resources for this object.

Parameters:

slot The slot the digital module is plugged into that corresponds to this serial port

channel The PWM channel for this PWM object

6.25.2 Variable Documentation

6.25.2.1 PWM* PWMs[SensorBase::kDigitalModules][SensorBase::kPwmChannels] [static]

6.25.2.2 bool PWMsInitialized = false [static]

6.26 CPWM.h File Reference

```
#include <VxWorks.h>
#include "CWrappers.h"
#include "PWM.h"
```

Functions

- PWM * AllocatePWM (UINT32 slot, UINT32 channel, SensorCreator creator)
- PWM * AllocatePWM (UINT32 channel, SensorCreator creator)
- void DeletePWM (UINT32 slot, UINT32 channel)

6.26.1 Function Documentation

6.26.1.1 PWM* AllocatePWM (UINT32 *channel*, SensorCreator *createObject*)

Alloate a PWM based object

Allocate an instance of a PWM based object. This code is shared between the subclasses of PWM and is not usually created as a standalone object.

Parameters:

channel The PWM channel for this PWM object

createObject The function callback in the subclass object that actually creates an instance of the appropriate class.

6.26.1.2 PWM* AllocatePWM (UINT32 *module*, UINT32 *channel*, SensorCreator *createObject*)

Alloate a PWM based object

Allocate an instance of a PWM based object. This code is shared between the subclasses of PWM and is not usually created as a standalone object.

Parameters:

module The slot the digital module is plugged into that corresponds to this serial port

channel The PWM channel for this PWM object

createObject The function callback in the subclass object that actually creates an instance of the appropriate class.

6.26.1.3 void DeletePWM (UINT32 *slot*, UINT32 *channel*)

Delete a PWM Delete a PWM and free up all the associated resources for this object.

Parameters:

slot The slot the digital module is plugged into that corresponds to this serial port

channel The PWM channel for this PWM object

6.27 CRelay.cpp File Reference

```
#include "SensorBase.h"
#include "DigitalModule.h"
#include "Relay.h"
#include "CRelay.h"
```

Functions

- static Relay * AllocateRelay (UINT32 slot, UINT32 channel)
- void InitRelay (UINT32 slot, UINT32 channel, RelayDirection direction)
- void InitRelay (UINT32 channel, RelayDirection direction)
- void DeleteRelay (UINT32 slot, UINT32 channel)
- void DeleteRelay (UINT32 channel)
- void SetRelay (UINT32 slot, UINT32 channel, RelayValue value)
- void SetRelay (UINT32 channel, RelayValue value)

Variables

- static Relay * relays [SensorBase::kDigitalModules][SensorBase::kRelayChannels]
- static bool initialized = false
- static Relay::Direction s_direction = Relay::kBothDirections

6.27.1 Function Documentation

6.27.1.1 static Relay* AllocateRelay (UINT32 *slot*, UINT32 *channel*) [static]

Internal function to allocate Relay objects. This function handles the mapping between channel/slot numbers to relay objects. It also allocates Relay objects if they are not already allocated.

Parameters:

slot The slot the digital module is plugged into

channel The relay channel for this device

6.27.1.2 void DeleteRelay (UINT32 *channel*)

Free up the resources associated with this relay. Delete the underlying Relay object and make the channel/port available for reuse.

Parameters:

channel The relay channel number for this object

6.27.1.3 void DeleteRelay (UINT32 *slot*, UINT32 *channel*)

Free up the resources associated with this relay. Delete the underlying Relay object and make the channel/port available for reuse.

Parameters:

slot The slot that the digital module is plugged into

channel The relay channel number for this object

6.27.1.4 void InitRelay (UINT32 *channel*, RelayDirection *direction*)

Set the direction that this relay object will control.

Parameters:

channel The relay channel number for this object

direction The direction that the relay object will control

6.27.1.5 void InitRelay (UINT32 *slot*, UINT32 *channel*, RelayDirection *direction*)

Set the direction that this relay object will control.

Parameters:

slot The slot the digital module is plugged into

channel The relay channel number for this object

direction The direction that the relay object will control

6.27.1.6 void SetRelay (UINT32 *channel*, RelayValue *value*)

Set the relay state.

Valid values depend on which directions of the relay are controlled by the object.

When set to kBothDirections, the relay can only be one of the three reasonable values, 0v-0v, 0v-12v, or 12v-0v.

When set to kForwardOnly or kReverseOnly, you can specify the constant for the direction or you can simply specify kOff and kOn. Using only kOff and kOn is recommended.

Parameters:

channel The relay channel number for this object

value The state to set the relay.

6.27.1.7 void SetRelay (UINT32 *slot*, UINT32 *channel*, RelayValue *value*)

Set the relay state.

Valid values depend on which directions of the relay are controlled by the object.

When set to kBothDirections, the relay can only be one of the three reasonable values, 0v-0v, 0v-12v, or 12v-0v.

When set to kForwardOnly or kReverseOnly, you can specify the constant for the direction or you can simply specify kOff and kOn. Using only kOff and kOn is recommended.

Parameters:

slot The slot that the digital module is plugged into

channel The relay channel number for this object

value The state to set the relay.

6.27.2 Variable Documentation

6.27.2.1 bool initialized = false [static]

6.27.2.2 Relay* relays[SensorBase::kDigitalModules][SensorBase::kRelayChannels] [static]

6.27.2.3 Relay::Direction s_direction = Relay::kBothDirections [static]

6.28 CRelay.h File Reference

Enumerations

- enum RelayValue { kOff, kOn, kForward, kReverse }
- enum RelayDirection { kBothDirections, kForwardOnly, kReverseOnly }

Functions

- void InitRelay (UINT32 channel, RelayDirection direction=kBothDirections)
- void InitRelayRelay (UINT32 slot, UINT32 channel, RelayDirection direction=kBothDirections)
- void DeleteRelay (UINT32 channel)
- void DeleteRelay (UINT32 slot, UINT32 channel)
- void SetRelay (UINT32 channel, RelayValue value)
- void SetRelay (UINT32 slot, UINT32 channel, RelayValue value)

6.28.1 Enumeration Type Documentation

6.28.1.1 enum RelayDirection

Enumerator:

kBothDirections

kForwardOnly

kReverseOnly

6.28.1.2 enum RelayValue

Enumerator:

kOff

kOn

kForward

kReverse

6.28.2 Function Documentation

6.28.2.1 void DeleteRelay (UINT32 *slot*, UINT32 *channel*)

Free up the resources associated with this relay. Delete the underlying Relay object and make the channel/port available for reuse.

Parameters:

slot The slot that the digital module is plugged into

channel The relay channel number for this object

6.28.2.2 void DeleteRelay (UINT32 *channel*)

Free up the resources associated with this relay. Delete the underlying Relay object and make the channel/port available for reuse.

Parameters:

channel The relay channel number for this object

6.28.2.3 void InitRelay (UINT32 *channel*, RelayDirection *direction*)

Set the direction that this relay object will control.

Parameters:

channel The relay channel number for this object

direction The direction that the relay object will control

6.28.2.4 void InitRelayRelay (UINT32 *slot*, UINT32 *channel*, RelayDirection *direction* = `kBothDirections`)

6.28.2.5 void SetRelay (UINT32 *slot*, UINT32 *channel*, RelayValue *value*)

Set the relay state.

Valid values depend on which directions of the relay are controlled by the object.

When set to kBothDirections, the relay can only be one of the three reasonable values, 0v-0v, 0v-12v, or 12v-0v.

When set to kForwardOnly or kReverseOnly, you can specify the constant for the direction or you can simply specify kOff and kOn. Using only kOff and kOn is recommended.

Parameters:

slot The slot that the digital module is plugged into

channel The relay channel number for this object

value The state to set the relay.

6.28.2.6 void SetRelay (UINT32 *channel*, RelayValue *value*)

Set the relay state.

Valid values depend on which directions of the relay are controlled by the object.

When set to kBothDirections, the relay can only be one of the three reasonable values, 0v-0v, 0v-12v, or 12v-0v.

When set to kForwardOnly or kReverseOnly, you can specify the constant for the direction or you can simply specify kOff and kOn. Using only kOff and kOn is recommended.

Parameters:

channel The relay channel number for this object

value The state to set the relay.

6.29 CRobotDrive.cpp File Reference

```
#include "CRobotDrive.h"
#include "Joystick.h"
#include "RobotDrive.h"
#include "Utility.h"
#include "WPIStatus.h"
```

Functions

- void CreateRobotDrive (UINT32 frontLeftMotor, UINT32 rearLeftMotor, UINT32 frontRightMotor, UINT32 rearRightMotor, float sensitivity)
- void CreateRobotDrive (UINT32 leftMotor, UINT32 rightMotor, float sensitivity)
- void Drive (float speed, float curve)
- void TankDrive (UINT32 leftStickPort, UINT32 rightStickPort)
- void ArcadeDrive (UINT32 stickPort, bool squaredInputs)
- void TankByValue (float leftSpeed, float rightSpeed)
- void ArcadeByValue (float moveValue, float rotateValue, bool squaredInputs)

Variables

- static RobotDrive * drive = NULL

6.29.1 Function Documentation

6.29.1.1 void ArcadeByValue (float *moveValue*, float *rotateValue*, bool *squaredInputs*)

Arcade drive implements single stick driving. This function lets you directly provide joystick values from any source.

Parameters:

moveValue The value to use for fowards/backwards

rotateValue The value to use for the rotate right/left

squaredInputs If set, increases the sensitivity at low speeds

6.29.1.2 void ArcadeDrive (UINT32 *stickPort*, bool *squaredInputs*)

Arcade drive implements single stick driving. Given a single Joystick, the class assumes the Y axis for the move value and the X axis for the rotate value. (Should add more information here regarding the way that arcade drive works.)

Parameters:

stickPort The joystick to use for Arcade single-stick driving. The Y-axis will be selected for forwards/backwards and the X-axis will be selected for rotation rate.

squaredInputs If true, the sensitivity will be increased for small values

6.29.1.3 void CreateRobotDrive (UINT32 *leftMotor*, UINT32 *rightMotor*, float *sensitivity*)

Constructor for RobotDrive with 2 motors specified with channel numbers. Set up parameters for a four wheel drive system where all four motor pwm channels are specified in the call. This call assumes Jaguars for controlling the motors.

Parameters:

leftMotor Front left motor channel number on the default digital module

rightMotor Front right motor channel number on the default digital module

sensitivity Effectively sets the turning sensitivity (or turn radius for a given value)

6.29.1.4 void CreateRobotDrive (UINT32 *frontLeftMotor*, UINT32 *rearLeftMotor*, UINT32 *frontRightMotor*, UINT32 *rearRightMotor*, float *sensitivity*)

Create a RobotDrive with 4 motors specified with channel numbers. Set up parameters for a four wheel drive system where all four motor pwm channels are specified in the call. This call assumes Jaguars for controlling the motors.

Parameters:

frontLeftMotor Front left motor channel number on the default digital module

rearLeftMotor Rear Left motor channel number on the default digital module

frontRightMotor Front right motor channel number on the default digital module

rearRightMotor Rear Right motor channel number on the default digital module

sensitivity Effectively sets the turning sensitivity (or turn radius for a given value)

6.29.1.5 void Drive (float *speed*, float *curve*)

Drive the motors at "speed" and "curve".

The speed and curve are -1.0 to +1.0 values where 0.0 represents stopped and not turning. The algorithm for adding in the direction attempts to provide a constant turn radius for differing speeds.

This function sill most likely be used in an autonomous routine.

Parameters:

speed The forward component of the speed to send to the motors.

curve The rate of turn, constant for different forward speeds.

6.29.1.6 void TankByValue (float *leftSpeed*, float *rightSpeed*)

Provide tank steering using the stored robot configuration. This function lets you directly provide joystick values from any source.

Parameters:

leftSpeed The value of the left stick.

rightSpeed The value of the right stick.

6.29.1.7 void TankDrive (UINT32 *leftStickPort*, UINT32 *rightStickPort*)

Provide tank steering using the stored robot configuration. Drive the robot using two joystick inputs. The Y-axis will be selected from each Joystick object.

Parameters:

leftStickPort The joystick port to control the left side of the robot.

rightStickPort The joystick port to control the right side of the robot.

6.29.2 Variable Documentation

6.29.2.1 RobotDrive* drive = NULL [static]

6.30 CRobotDrive.h File Reference

```
#include <VxWorks.h>
```

Functions

- void CreateRobotDrive (UINT32 leftMotor, UINT32 rightMotor, float sensitivity=0.5)
- void CreateRobotDrive (UINT32 frontLeftMotor, UINT32 rearLeftMotor, UINT32 frontRightMotor, UINT32 rearRightMotor, float sensitivity=0.5)
- void Drive (float speed, float curve)
- void TankDrive (UINT32 leftStickPort, UINT32 rightStickPort)
- void ArcadeDrive (UINT32 stickPort, bool squaredInputs=false)
- void TankByValue (float leftSpeed, float rightSpeed)
- void ArcadeByValue (float moveSpeed, float rotateSpeed, bool squaredInputs=false)

6.30.1 Function Documentation

6.30.1.1 void ArcadeByValue (float *moveValue*, float *rotateValue*, bool *squaredInputs*)

Arcade drive implements single stick driving. This function lets you directly provide joystick values from any source.

Parameters:

moveValue The value to use for fowards/backwards

rotateValue The value to use for the rotate right/left

squaredInputs If set, increases the sensitivity at low speeds

6.30.1.2 void ArcadeDrive (UINT32 *stickPort*, bool *squaredInputs*)

Arcade drive implements single stick driving. Given a single Joystick, the class assumes the Y axis for the move value and the X axis for the rotate value. (Should add more information here regarding the way that arcade drive works.)

Parameters:

stickPort The joystick to use for Arcade single-stick driving. The Y-axis will be selected for forwards/backwards and the X-axis will be selected for rotation rate.

squaredInputs If true, the sensitivity will be increased for small values

6.30.1.3 void CreateRobotDrive (UINT32 *frontLeftMotor*, UINT32 *rearLeftMotor*, UINT32 *frontRightMotor*, UINT32 *rearRightMotor*, float *sensitivity*)

Create a RobotDrive with 4 motors specified with channel numbers. Set up parameters for a four wheel drive system where all four motor pwm channels are specified in the call. This call assumes Jaguars for controlling the motors.

Parameters:

frontLeftMotor Front left motor channel number on the default digital module

rearLeftMotor Rear Left motor channel number on the default digital module

frontRightMotor Front right motor channel number on the default digital module

rearRightMotor Rear Right motor channel number on the default digital module

sensitivity Effectively sets the turning sensitivity (or turn radius for a given value)

6.30.1.4 void CreateRobotDrive (UINT32 *leftMotor*, UINT32 *rightMotor*, float *sensitivity*)

Constructor for RobotDrive with 2 motors specified with channel numbers. Set up parameters for a four wheel drive system where all four motor pwm channels are specified in the call. This call assumes Jaguars for controlling the motors.

Parameters:

leftMotor Front left motor channel number on the default digital module

rightMotor Front right motor channel number on the default digital module

sensitivity Effectively sets the turning sensitivity (or turn radius for a given value)

6.30.1.5 void Drive (float *speed*, float *curve*)

Drive the motors at "speed" and "curve".

The speed and curve are -1.0 to +1.0 values where 0.0 represents stopped and not turning. The algorithm for adding in the direction attempts to provide a constant turn radius for differing speeds.

This function sill most likely be used in an autonomous routine.

Parameters:

speed The forward component of the speed to send to the motors.

curve The rate of turn, constant for different forward speeds.

6.30.1.6 void TankByValue (float *leftSpeed*, float *rightSpeed*)

Provide tank steering using the stored robot configuration. This function lets you directly provide joystick values from any source.

Parameters:

leftSpeed The value of the left stick.

rightSpeed The value of the right stick.

6.30.1.7 void TankDrive (UINT32 *leftStickPort*, UINT32 *rightStickPort*)

Provide tank steering using the stored robot configuration. Drive the robot using two joystick inputs. The Y-axis will be selected from each Joystick object.

Parameters:

leftStickPort The joystick port to control the left side of the robot.

rightStickPort The joystick port to control the right side of the robot.

6.31 CSerialPort.cpp File Reference

```
#include "CSerialPort.h"
#include <visa/visa.h>
```

Functions

- void OpenSerialPort (UINT32 baudRate, UINT8 dataBits, SerialPort::Parity parity, SerialPort::StopBits stopBits)
- void SetSerialFlowControl (SerialPort::FlowControl flowControl)
- void EnableSerialTermination (char terminator)
- void DisableSerialTermination (void)
- INT32 GetSerialBytesReceived (void)
- void PrintfSerial (const char *writeFmt,...)
- void ScanfSerial (const char *readFmt,...)
- UINT32 ReadSerialPort (char *buffer, INT32 count)
- UINT32 WriteSerialPort (const char *buffer, INT32 count)
- void SetSerialTimeout (INT32 timeout)
- void SetSerialWriteBufferMode (SerialPort::WriteBufferMode mode)
- void FlushSerialPort (void)
- void ResetSerialPort (void)

Variables

- static SerialPort * serial_port = NULL

6.31.1 Function Documentation

6.31.1.1 void DisableSerialTermination (void)

Disable termination behavior.

6.31.1.2 void EnableSerialTermination (char *terminator*)

Enable termination and specify the termination character.

Termination is currently only implemented for receive. When the the terminator is recieved, the Read() or Scanf() will return fewer bytes than requested, stopping after the terminator.

Parameters:

terminator The character to use for termination.

6.31.1.3 void FlushSerialPort (void)

Force the output buffer to be written to the port.

This is used when SetWriteBufferMode() is set to kFlushWhenFull to force a flush before the buffer is full.

6.31.1.4 INT32 GetSerialBytesReceived (void)

Get the number of bytes currently available to read from the serial port.

Returns:

The number of bytes available to read.

6.31.1.5 void OpenSerialPort (UINT32 *baudRate*, UINT8 *dataBits*, SerialPort::Parity *parity*, SerialPort::StopBits *stopBits*)

Open the serial port object. Open and allocate the serial port.

Parameters:

baudRate The baud rate to configure the serial port. The cRIO-9074 supports up to 230400 Baud.

dataBits The number of data bits per transfer. Valid values are between 5 and 8 bits.

parity Select the type of parity checking to use.

stopBits The number of stop bits to use as defined by the enum StopBits.

6.31.1.6 void PrintfSerial (const char * *writeFmt*, ...)

Output formatted text to the serial port.

Bug

All pointer-based parameters seem to return an error.

Parameters:

writeFmt A string that defines the format of the output.

6.31.1.7 UINT32 ReadSerialPort (char * *buffer*, INT32 *count*)

Read raw bytes out of the buffer.

Parameters:

buffer Pointer to the buffer to store the bytes in.

count The maximum number of bytes to read.

Returns:

The number of bytes actually read into the buffer.

6.31.1.8 void ResetSerialPort (void)

Reset the serial port driver to a known state.

Empty the transmit and receive buffers in the device and formatted I/O.

6.31.1.9 void ScanfSerial (const char * *readFmt*, ...)

Input formatted text from the serial port.

Bug

All pointer-based parameters seem to return an error.

Parameters:

readFmt A string that defines the format of the input.

6.31.1.10 void SetSerialFlowControl (SerialPort::FlowControl *flowControl*)

Set the type of flow control to enable on this port.

By default, flow control is disabled.

6.31.1.11 void SetSerialTimeout (INT32 *timeout*)

Configure the timeout of the serial port.

This defines the timeout for transactions with the hardware. It will affect reads and very large writes.

Parameters:

timeout The number of seconds to to wait for I/O.

6.31.1.12 void SetSerialWriteBufferMode (SerialPort::WriteBufferMode *mode*)

Specify the flushing behavior of the output buffer.

When set to kFlushOnAccess, data is synchronously written to the serial port after each call to either Printf() or Write().

When set to kFlushWhenFull, data will only be written to the serial port when the buffer is full or when Flush() is called.

Parameters:

mode The write buffer mode.

6.31.1.13 UINT32 WriteSerialPort (const char * *buffer*, INT32 *count*)

Write raw bytes to the buffer.

Parameters:

buffer Pointer to the buffer to read the bytes from.

count The maximum number of bytes to write.

Returns:

The number of bytes actually written into the port.

6.31.2 Variable Documentation

6.31.2.1 SerialPort* serial_port = NULL [static]

6.32 CSerialPort.h File Reference

```
#include "SerialPort.h"
```

Functions

- void OpenSerialPort (UINT32 baudRate, UINT8 dataBits, SerialPort::Parity parity, SerialPort::StopBits stopBits)
- void SetSerialFlowControl (SerialPort::FlowControl flowControl)
- void EnableSerialTermination (char terminator)
- void DisableSerialTermination (void)
- INT32 GetSerialBytesReceived (void)
- void PrintfSerial (const char *writeFmt,...)
- void ScanfSerial (const char *readFmt,...)
- UINT32 ReadSerialPort (char *buffer, INT32 count)
- UINT32 WriteSerialPort (const char *buffer, INT32 count)
- void SetSerialTimeout (INT32 timeout_ms)
- void SetSerialWriteBufferMode (SerialPort::WriteBufferMode mode)
- void FlushSerialPort (void)
- void ResetSerialPort (void)

6.32.1 Function Documentation

6.32.1.1 void DisableSerialTermination (void)

Disable termination behavior.

6.32.1.2 void EnableSerialTermination (char *terminator*)

Enable termination and specify the termination character.

Termination is currently only implemented for receive. When the the terminator is recieved, the Read() or Scanf() will return fewer bytes than requested, stopping after the terminator.

Parameters:

terminator The character to use for termination.

6.32.1.3 void FlushSerialPort (void)

Force the output buffer to be written to the port.

This is used when SetWriteBufferMode() is set to kFlushWhenFull to force a flush before the buffer is full.

6.32.1.4 INT32 GetSerialBytesReceived (void)

Get the number of bytes currently available to read from the serial port.

Returns:

The number of bytes available to read.

6.32.1.5 void OpenSerialPort (UINT32 *baudRate*, UINT8 *dataBits*, SerialPort::Parity *parity*, SerialPort::StopBits *stopBits*)

Open the serial port object. Open and allocate the serial port.

Parameters:

baudRate The baud rate to configure the serial port. The cRIO-9074 supports up to 230400 Baud.

dataBits The number of data bits per transfer. Valid values are between 5 and 8 bits.

parity Select the type of parity checking to use.

stopBits The number of stop bits to use as defined by the enum StopBits.

6.32.1.6 void PrintfSerial (const char * *writeFmt*, ...)

Output formatted text to the serial port.

Bug

All pointer-based parameters seem to return an error.

Parameters:

writeFmt A string that defines the format of the output.

6.32.1.7 UINT32 ReadSerialPort (char * *buffer*, INT32 *count*)

Read raw bytes out of the buffer.

Parameters:

buffer Pointer to the buffer to store the bytes in.

count The maximum number of bytes to read.

Returns:

The number of bytes actually read into the buffer.

6.32.1.8 void ResetSerialPort (void)

Reset the serial port driver to a known state.

Empty the transmit and receive buffers in the device and formatted I/O.

6.32.1.9 void ScanfSerial (const char * *readFmt*, ...)

Input formatted text from the serial port.

Bug

All pointer-based parameters seem to return an error.

Parameters:

readFmt A string that defines the format of the input.

6.32.1.10 void SetSerialFlowControl (SerialPort::FlowControl *flowControl*)

Set the type of flow control to enable on this port.

By default, flow control is disabled.

6.32.1.11 void SetSerialTimeout (INT32 *timeout*)

Configure the timeout of the serial port.

This defines the timeout for transactions with the hardware. It will affect reads and very large writes.

Parameters:

timeout The number of seconds to to wait for I/O.

6.32.1.12 void SetSerialWriteBufferMode (SerialPort::WriteBufferMode *mode*)

Specify the flushing behavior of the output buffer.

When set to kFlushOnAccess, data is synchronously written to the serial port after each call to either Printf() or Write().

When set to kFlushWhenFull, data will only be written to the serial port when the buffer is full or when Flush() is called.

Parameters:

mode The write buffer mode.

6.32.1.13 UINT32 WriteSerialPort (const char * *buffer*, INT32 *count*)

Write raw bytes to the buffer.

Parameters:

buffer Pointer to the buffer to read the bytes from.

count The maximum number of bytes to write.

Returns:

The number of bytes actually written into the port.

6.33 CServo.cpp File Reference

```
#include "Servo.h"
#include "CServo.h"
#include "CPWM.h"
```

Functions

- static SensorBase * CreateServo (UINT32 slot, UINT32 channel)
- void SetServo (UINT32 slot, UINT32 channel, float value)
- float GetGetServo (UINT32 slot, UINT32 channel)
- void SetServoAngle (UINT32 slot, UINT32 channel, float angle)
- float GetServoAngle (UINT32 slot, UINT32 channel)
- float GetServoMaxAngle (UINT32 slot, UINT32 channel)
- float GetServoMinAngle (UINT32 slot, UINT32 channel)
- void SetServo (UINT32 channel, float value)
- float GetServo (UINT32 channel)
- void SetServoAngle (UINT32 channel, float angle)
- float GetServoAngle (UINT32 channel)
- float GetServoMaxAngle (UINT32 channel)
- float GetServoMinAngle (UINT32 channel)
- void DeleteServo (UINT32 slot, UINT32 channel)
- void DeleteServo (UINT32 channel)

6.33.1 Function Documentation

6.33.1.1 static SensorBase* CreateServo (UINT32 *slot*, UINT32 *channel*) [static]

6.33.1.2 void DeleteServo (UINT32 *channel*)

Free the resources associated with this Servo object. The underlying Servo object and the allocated ports are freed.

Parameters:

channel The PWM port in the module the servo is plugged into

6.33.1.3 void DeleteServo (UINT32 *slot*, UINT32 *channel*)

Free the resources associated with this Servo object. The underlying Servo object and the allocated ports are freed.

Parameters:

slot The slot the digital module is plugged into

channel The PWM port in the module the servo is plugged into

6.33.1.4 float GetGetServo (UINT32 *slot*, UINT32 *channel*)

Get the servo position.

Servo values range from 0.0 to 1.0 corresponding to the range of full left to full right.

Parameters:

slot The slot the digital module is plugged into
channel The PWM channel in the module the servo is plugged into

Returns:

Position from 0.0 to 1.0.

6.33.1.5 float GetServo (UINT32 *channel*)

Get the servo position.

Servo values range from 0.0 to 1.0 corresponding to the range of full left to full right.

Parameters:

channel The PWM port in the module the servo is plugged into

Returns:

Position from 0.0 to 1.0.

6.33.1.6 float GetServoAngle (UINT32 *channel*)

Get the servo angle.

Assume that the servo angle is linear with respect to the PWM value (big assumption, need to test).

Parameters:

channel The slot the digital module is plugged into

Returns:

The angle in degrees to which the servo is set.

6.33.1.7 float GetServoAngle (UINT32 *slot*, UINT32 *channel*)

Get the servo angle.

Assume that the servo angle is linear with respect to the PWM value (big assumption, need to test).

Parameters:

slot The slot the digital module is plugged into
channel The PWM channel in the module the servo is plugged into

Returns:

The angle in degrees to which the servo is set.

6.33.1.8 float GetServoMaxAngle (UINT32 *channel*)

Get the maximum angle for the servo.

Parameters:

channel The PWM port in the module the servo is plugged into

6.33.1.9 float GetServoMaxAngle (UINT32 *slot*, UINT32 *channel*)

Get the maximum servo angle.

Parameters:

slot The slot the digital module is plugged into

channel The PWM port in the module the servo is plugged into

6.33.1.10 float GetServoMinAngle (UINT32 *channel*)

Get the minimum angle for the servo.

Parameters:

channel The PWM port in the module the servo is plugged into

6.33.1.11 float GetServoMinAngle (UINT32 *slot*, UINT32 *channel*)

Get the minimum servo angle.

Parameters:

slot The slot the digital module is plugged into

channel The PWM port in the module the servo is plugged into

6.33.1.12 void SetServo (UINT32 *channel*, float *value*)

Set the servo position.

Servo values range from 0.0 to 1.0 corresponding to the range of full left to full right.

Parameters:

channel The PWM port in the module the servo is plugged into

value Position from 0.0 to 1.0.

6.33.1.13 void SetServo (UINT32 *slot*, UINT32 *channel*, float *value*)

Set the servo position.

Servo values range from 0.0 to 1.0 corresponding to the range of full left to full right.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel in the module the servo is plugged into

value Position from 0.0 to 1.0.

6.33.1.14 void SetServoAngle (UINT32 *channel*, float *angle*)

Set the servo angle.

Assume that the servo angle is linear with respect to the PWM value (big assumption, need to test).

Servo angles that are out of the supported range of the servo simply "saturate" in that direction In other words, if the servo has a range of (X degrees to Y degrees) than angles of less than X result in an angle of X being set and angles of more than Y degrees result in an angle of Y being set.

Parameters:

channel The PWM port in the module the servo is plugged into

angle The angle in degrees to set the servo.

6.33.1.15 void SetServoAngle (UINT32 *slot*, UINT32 *channel*, float *angle*)

Set the servo angle.

Assume that the servo angle is linear with respect to the PWM value (big assumption, need to test).

Servo angles that are out of the supported range of the servo simply "saturate" in that direction In other words, if the servo has a range of (X degrees to Y degrees) than angles of less than X result in an angle of X being set and angles of more than Y degrees result in an angle of Y being set.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel in the module the servo is plugged into

angle The angle in degrees to set the servo.

6.34 CServo.h File Reference

Functions

- void SetServo (UINT32 slot, UINT32 channel, float value)
- float GetGetServo (UINT32 slot, UINT32 channel)
- void SetServoAngle (UINT32 slot, UINT32 channel, float angle)
- float GetServoAngle (UINT32 slot, UINT32 channel)
- float GetServoMaxAngle (UINT32 slot, UINT32 channel)
- float GetServoMinAngle (UINT32 slot, UINT32 channel)
- void SetServo (UINT32 channel, float value)
- float GetGetServo (UINT32 channel)
- void SetServoAngle (UINT32 channel, float angle)
- float GetServoAngle (UINT32 channel)
- float GetServoMaxAngle (UINT32 channel)
- float GetServoMinAngle (UINT32 channel)
- void DeleteServo (UINT32 slot, UINT32 channel)
- void DeleteServo (UINT32 channel)

6.34.1 Function Documentation

6.34.1.1 void DeleteServo (UINT32 *channel*)

Free the resources associated with this Servo object. The underlying Servo object and the allocated ports are freed.

Parameters:

channel The PWM port in the module the servo is plugged into

6.34.1.2 void DeleteServo (UINT32 *slot*, UINT32 *channel*)

Free the resources associated with this Servo object. The underlying Servo object and the allocated ports are freed.

Parameters:

slot The slot the digital module is plugged into

channel The PWM port in the module the servo is plugged into

6.34.1.3 float GetGetServo (UINT32 *channel*)

6.34.1.4 float GetGetServo (UINT32 *slot*, UINT32 *channel*)

Get the servo position.

Servo values range from 0.0 to 1.0 corresponding to the range of full left to full right.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel in the module the servo is plugged into

Returns:

Position from 0.0 to 1.0.

6.34.1.5 float GetServoAngle (UINT32 *channel*)

Get the servo angle.

Assume that the servo angle is linear with respect to the PWM value (big assumption, need to test).

Parameters:

channel The slot the digital module is plugged into

Returns:

The angle in degrees to which the servo is set.

6.34.1.6 float GetServoAngle (UINT32 *slot*, UINT32 *channel*)

Get the servo angle.

Assume that the servo angle is linear with respect to the PWM value (big assumption, need to test).

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel in the module the servo is plugged into

Returns:

The angle in degrees to which the servo is set.

6.34.1.7 float GetServoMaxAngle (UINT32 *channel*)

Get the maximum angle for the servo.

Parameters:

channel The PWM port in the module the servo is plugged into

6.34.1.8 float GetServoMaxAngle (UINT32 *slot*, UINT32 *channel*)

Get the maximum servo angle.

Parameters:

slot The slot the digital module is plugged into

channel The PWM port in the module the servo is plugged into

6.34.1.9 float GetServoMinAngle (UINT32 *channel*)

Get the minimum angle for the servo.

Parameters:

channel The PWM port in the module the servo is plugged into

6.34.1.10 float GetServoMinAngle (UINT32 *slot*, UINT32 *channel*)

Get the minimum servo angle.

Parameters:

slot The slot the digital module is plugged into
channel The PWM port in the module the servo is plugged into

6.34.1.11 void SetServo (UINT32 *channel*, float *value*)

Set the servo position.

Servo values range from 0.0 to 1.0 corresponding to the range of full left to full right.

Parameters:

channel The PWM port in the module the servo is plugged into
value Position from 0.0 to 1.0.

6.34.1.12 void SetServo (UINT32 *slot*, UINT32 *channel*, float *value*)

Set the servo position.

Servo values range from 0.0 to 1.0 corresponding to the range of full left to full right.

Parameters:

slot The slot the digital module is plugged into
channel The PWM channel in the module the servo is plugged into
value Position from 0.0 to 1.0.

6.34.1.13 void SetServoAngle (UINT32 *channel*, float *angle*)

Set the servo angle.

Assume that the servo angle is linear with respect to the PWM value (big assumption, need to test).

Servo angles that are out of the supported range of the servo simply "saturate" in that direction In other words, if the servo has a range of (X degrees to Y degrees) than angles of less than X result in an angle of X being set and angles of more than Y degrees result in an angle of Y being set.

Parameters:

channel The PWM port in the module the servo is plugged into
angle The angle in degrees to set the servo.

6.34.1.14 void SetServoAngle (UINT32 *slot*, UINT32 *channel*, float *angle*)

Set the servo angle.

Assume that the servo angle is linear with respect to the PWM value (big assumption, need to test).

Servo angles that are out of the supported range of the servo simply "saturate" in that direction In other words, if the servo has a range of (X degrees to Y degrees) than angles of less than X result in an angle of X being set and angles of more than Y degrees result in an angle of Y being set.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel in the module the servo is plugged into

angle The angle in degrees to set the servo.

6.35 CSolenoid.cpp File Reference

```
#include "Solenoid.h"
#include "CSolenoid.h"
```

Functions

- static Solenoid * allocateSolenoid (UINT32 channel)
- void SetSolenoid (UINT32 channel, bool on)
- bool GetSolenoid (UINT32 channel)
- void DeleteSolenoid (UINT32 channel)

Variables

- static Solenoid * solenoids [SensorBase::kSolenoidChannels]
- static bool initialized = false

6.35.1 Function Documentation

6.35.1.1 static Solenoid* allocateSolenoid (UINT32 *channel*) [static]

Internal allocation function for Solenoid channels. The function is used interally to allocate the Solenoid object and keep track of the channel mapping to the object for subsequent calls.

Parameters:

channel The channel for the solenoid

6.35.1.2 void DeleteSolenoid (UINT32 *channel*)

Free the resources associated with the Solenoid channel. Free the resources including the Solenoid object for this channel.

Parameters:

channel The channel in the Solenoid module

6.35.1.3 bool GetSolenoid (UINT32 *channel*)

Read the current value of the solenoid.

Parameters:

channel The channel in the Solenoid module

Returns:

The current value of the solenoid.

6.35.1.4 void SetSolenoid (UINT32 *channel*, bool *on*)

Set the value of a solenoid.

Parameters:

channel The channel on the Solenoid module

on Turn the solenoid output off or on.

6.35.2 Variable Documentation

6.35.2.1 bool initialized = false [static]

6.35.2.2 Solenoid* solenoids[SensorBase::kSolenoidChannels] [static]

6.36 CSolenoid.h File Reference

Functions

- void SetSolenoid (UINT32 channel, bool on)
- bool GetSolenoid (UINT32 channel)

6.36.1 Function Documentation

6.36.1.1 bool GetSolenoid (UINT32 *channel*)

Read the current value of the solenoid.

Parameters:

channel The channel in the Solenoid module

Returns:

The current value of the solenoid.

6.36.1.2 void SetSolenoid (UINT32 *channel*, bool *on*)

Set the value of a solenoid.

Parameters:

channel The channel on the Solenoid module

on Turn the solenoid output off or on.

6.37 CTimer.cpp File Reference

```
#include "CTimer.h"
#include <stdio.h>
#include "Utility.h"
```

Functions

- static Timer * AllocateTimer (UINT32 index)
- void ResetTimer (UINT32 index)
- void StartTimer (UINT32 index)
- void StopTimer (UINT32 index)
- double GetTimer (UINT32 index)
- void DeleteTimer (UINT32 index)

Variables

- static Timer * timers [kMaxTimers]
- static bool initialized = false

6.37.1 Function Documentation

6.37.1.1 static Timer* AllocateTimer (UINT32 *index*) [static]

Allocate the resources for a timer object Timers are allocated in an array and indexed with the "index" parameter. There can be up to 10 timer objects in use at any one time. Deleting a timer object frees up it's slot and resources.

Parameters:

index The index of this timer object.

6.37.1.2 void DeleteTimer (UINT32 *index*)

Free the resources associated with this timer object

Parameters:

index The index of this timer object.

6.37.1.3 double GetTimer (UINT32 *index*)

Get the current time from the timer. If the clock is running it is derived from the current system clock the start time stored in the timer class. If the clock is not running, then return the time when it was last stopped.

Parameters:

index The timer index being used

Returns:

unsigned Current time value for this timer in seconds

6.37.1.4 void ResetTimer (UINT32 *index*)

Reset the timer by setting the time to 0.

Make the timer startTime the current time so new requests will be relative now

Parameters:

index The index of this timer object.

6.37.1.5 void StartTimer (UINT32 *index*)

Start the timer running. Just set the running flag to true indicating that all time requests should be relative to the system clock.

Parameters:

index The index of this timer object.

6.37.1.6 void StopTimer (UINT32 *index*)

Stop the timer. This computes the time as of now and clears the running flag, causing all subsequent time requests to be read from the accumulated time rather than looking at the system clock.

Parameters:

index The index of this timer object.

6.37.2 Variable Documentation

6.37.2.1 bool initialized = false [static]

6.37.2.2 Timer* timers[kMaxTimers] [static]

6.38 CTimer.h File Reference

```
#include "Timer.h"
```

Functions

- void ResetTimer (UINT32 index)
- void StartTimer (UINT32 index)
- void StopTimer (UINT32 index)
- double GetTimer (UINT32 index)
- void DeleteTimer (UINT32 index)

Variables

- static const unsigned kMaxTimers = 32

6.38.1 Function Documentation

6.38.1.1 void DeleteTimer (UINT32 *index*)

Free the resources associated with this timer object

Parameters:

index The index of this timer object.

6.38.1.2 double GetTimer (UINT32 *index*)

Get the current time from the timer. If the clock is running it is derived from the current system clock the start time stored in the timer class. If the clock is not running, then return the time when it was last stopped.

Parameters:

index The timer index being used

Returns:

unsigned Current time value for this timer in seconds

6.38.1.3 void ResetTimer (UINT32 *index*)

Reset the timer by setting the time to 0.

Make the timer startTime the current time so new requests will be relative now

Parameters:

index The index of this timer object.

6.38.1.4 void StartTimer (UINT32 *index*)

Start the timer running. Just set the running flag to true indicating that all time requests should be relative to the system clock.

Parameters:

index The index of this timer object.

6.38.1.5 void StopTimer (UINT32 *index*)

Stop the timer. This computes the time as of now and clears the running flag, causing all subsequent time requests to be read from the accumulated time rather than looking at the system clock.

Parameters:

index The index of this timer object.

6.38.2 Variable Documentation

6.38.2.1 const unsigned kMaxTimers = 32 [static]

6.39 CUltrasonic.cpp File Reference

```
#include "CUltrasonic.h"
#include "DigitalModule.h"
```

Functions

- static void USinit (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)
- static Ultrasonic * USptr (UINT32 pingSlot, UINT32 pingChannel)
- void InitUltrasonic (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)
- void InitUltrasonic (UINT32 pingChannel, UINT32 echoChannel)
- double GetUltrasonicInches (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)
- double GetUltrasonicInches (UINT32 pingChannel, UINT32 echoChannel)
- double GetUltrasonicMM (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)
- double GetUltrasonicMM (UINT32 pingChannel, UINT32 echoChannel)
- void DeleteUltrasonic (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)
- void DeleteUltrasonic (UINT32 pingChannel, UINT32 echoChannel)

Variables

- static Ultrasonic * ultrasonics [SensorBase::kChassisSlots][SensorBase::kDigitalChannels]
- static bool initialized = false

6.39.1 Function Documentation

6.39.1.1 void DeleteUltrasonic (UINT32 *pingChannel*, UINT32 *echoChannel*)

Free the resources associated with an ultrasonic sensor. Deallocate the Ultrasonic object and free the associated resources.

Parameters:

pingChannel The channel on the digital module for the ping connection

echoChannel The channel on the digital module for the echo connection

6.39.1.2 void DeleteUltrasonic (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*)

Free the resources associated with an ultrasonic sensor. Deallocate the Ultrasonic object and free the associated resources.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.39.1.3 double GetUltrasonicInches (UINT32 *pingChannel*, UINT32 *echoChannel*)

Get the range in inches from the ultrasonic sensor.

Returns:

double Range in inches of the target returned from the ultrasonic sensor. If there is no valid value yet, i.e. at least one measurement hasn't completed, then return 0.

Parameters:

pingChannel The channel on the digital module for the ping connection

echoChannel The channel on the digital module for the echo connection

6.39.1.4 double GetUltrasonicInches (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*)

Get the range in inches from the ultrasonic sensor.

Returns:

double Range in inches of the target returned from the ultrasonic sensor. If there is no valid value yet, i.e. at least one measurement hasn't completed, then return 0.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.39.1.5 double GetUltrasonicMM (UINT32 *pingChannel*, UINT32 *echoChannel*)

Get the range in millimeters from the ultrasonic sensor.

Returns:

double Range in millimeters of the target returned by the ultrasonic sensor. If there is no valid value yet, i.e. at least one measurement hasn't complted, then return 0.

Parameters:

pingChannel The channel on the digital module for the ping connection

echoChannel The channel on the digital module for the echo connection

6.39.1.6 double GetUltrasonicMM (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*)

Get the range in millimeters from the ultrasonic sensor.

Returns:

double Range in millimeters of the target returned by the ultrasonic sensor. If there is no valid value yet, i.e. at least one measurement hasn't complted, then return 0.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.39.1.7 void InitUltrasonic (UINT32 *pingChannel*, UINT32 *echoChannel*)

Initialize and Ultrasonic sensor.

Initialize an Ultrasonic sensor to start it pinging in round robin mode with other allocated sensors. There is no need to explicitly start the sensor pinging.

Parameters:

pingChannel The channel on the digital module for the ping connection

echoChannel The channel on the digital module for the echo connection

6.39.1.8 void InitUltrasonic (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*)

Initialize and Ultrasonic sensor.

Initialize an Ultrasonic sensor to start it pinging in round robin mode with other allocated sensors. There is no need to explicitly start the sensor pinging.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.39.1.9 static void USinit (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*) [static]

Internal routine to allocate and initialize resources for an Ultrasonic sensor Allocate the actual Ultrasonic sensor object and the slot/channels associated with them. Then initialize the sensor.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.39.1.10 static Ultrasonic* USptr (UINT32 *pingSlot*, UINT32 *pingChannel*) [static]

Internal routine to return the pointer to an Ultrasonic sensor Return the pointer to a previously allocated Ultrasonic sensor object. Only the ping connection is required since there can only be a single sensor connected to that channel

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

6.39.2 Variable Documentation

6.39.2.1 bool initialized = false [static]

6.39.2.2 Ultrasonic* ultrasonics[SensorBase::kChassisSlots][SensorBase::kDigitalChannels] [static]

6.40 CUltrasonic.h File Reference

```
#include "Ultrasonic.h"
```

Functions

- void InitUltrasonic (UINT32 pingChannel, UINT32 echoChannel)
- void InitUltrasonic (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)
- double GetUltrasonicInches (UINT32 pingChannel, UINT32 echoChannel)
- double GetUltrasonicInches (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)
- double GetUltrasonicMM (UINT32 pingChannel, UINT32 echoChannel)
- double GetUltrasonicMM (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)
- void DeleteUltrasonic (UINT32 pingChannel, UINT32 echoChannel)
- void DeleteUltrasonic (UINT32 pingSlot, UINT32 pingChannel, UINT32 echoSlot, UINT32 echoChannel)

6.40.1 Function Documentation

6.40.1.1 void DeleteUltrasonic (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*)

Free the resources associated with an ultrasonic sensor. Deallocate the Ultrasonic object and free the associated resources.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.40.1.2 void DeleteUltrasonic (UINT32 *pingChannel*, UINT32 *echoChannel*)

Free the resources associated with an ultrasonic sensor. Deallocate the Ultrasonic object and free the associated resources.

Parameters:

pingChannel The channel on the digital module for the ping connection

echoChannel The channel on the digital module for the echo connection

6.40.1.3 double GetUltrasonicInches (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*)

Get the range in inches from the ultrasonic sensor.

Returns:

double Range in inches of the target returned from the ultrasonic sensor. If there is no valid value yet, i.e. at least one measurement hasn't completed, then return 0.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.40.1.4 double GetUltrasonicInches (UINT32 *pingChannel*, UINT32 *echoChannel*)

Get the range in inches from the ultrasonic sensor.

Returns:

double Range in inches of the target returned from the ultrasonic sensor. If there is no valid value yet, i.e. at least one measurement hasn't completed, then return 0.

Parameters:

pingChannel The channel on the digital module for the ping connection

echoChannel The channel on the digital module for the echo connection

6.40.1.5 double GetUltrasonicMM (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*)

Get the range in millimeters from the ultrasonic sensor.

Returns:

double Range in millimeters of the target returned by the ultrasonic sensor. If there is no valid value yet, i.e. at least one measurement hasn't complted, then return 0.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.40.1.6 double GetUltrasonicMM (UINT32 *pingChannel*, UINT32 *echoChannel*)

Get the range in millimeters from the ultrasonic sensor.

Returns:

double Range in millimeters of the target returned by the ultrasonic sensor. If there is no valid value yet, i.e. at least one measurement hasn't complted, then return 0.

Parameters:

pingChannel The channel on the digital module for the ping connection

echoChannel The channel on the digital module for the echo connection

6.40.1.7 void InitUltrasonic (UINT32 *pingSlot*, UINT32 *pingChannel*, UINT32 *echoSlot*, UINT32 *echoChannel*)

Initialize and Ultrasonic sensor.

Initialize an Ultrasonic sensor to start it pinging in round robin mode with other allocated sensors. There is no need to explicitly start the sensor pinging.

Parameters:

pingSlot The slot for the digital module for the ping connection

pingChannel The channel on the digital module for the ping connection

echoSlot The slot for the digital module for the echo connection

echoChannel The channel on the digital module for the echo connection

6.40.1.8 void InitUltrasonic (UINT32 *pingChannel*, UINT32 *echoChannel*)

Initialize and Ultrasonic sensor.

Initialize an Ultrasonic sensor to start it pinging in round robin mode with other allocated sensors. There is no need to explicitly start the sensor pinging.

Parameters:

pingChannel The channel on the digital module for the ping connection

echoChannel The channel on the digital module for the echo connection

6.41 CVictor.cpp File Reference

```
#include "CVictor.h"
#include "CPWM.h"
#include "Victor.h"
```

Functions

- static SensorBase * CreateVictor (UINT32 slot, UINT32 channel)
- void SetVictorSpeed (UINT32 slot, UINT32 channel, float speed)
- void SetVictorRaw (UINT32 channel, UINT8 value)
- void SetVictorSpeed (UINT32 channel, float speed)
- UINT8 GetVictorRaw (UINT32 channel)
- void SetVictorRaw (UINT32 slot, UINT32 channel, UINT8 value)
- UINT8 GetVictorRaw (UINT32 slot, UINT32 channel)
- void DeleteVictor (UINT32 slot, UINT32 channel)
- void DeleteVictor (UINT32 channel)

6.41.1 Function Documentation

6.41.1.1 static SensorBase* CreateVictor (UINT32 *slot*, UINT32 *channel*) [static]

Create an instance of a Victor object (used internally by this module)

Parameters:

slot The slot that the digital module is plugged into

channel The PWM channel that the motor is plugged into

6.41.1.2 void DeleteVictor (UINT32 *channel*)

Delete resources for a Victor Free the underlying object and delete the allocated resources for the Victor

Parameters:

channel The PWM channel used for this Victor

6.41.1.3 void DeleteVictor (UINT32 *slot*, UINT32 *channel*)

Delete resources for a Victor Free the underlying object and delete the allocated resources for the Victor

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel used for this Victor

6.41.1.4 UINT8 GetVictorRaw (UINT32 *slot*, UINT32 *channel*)

Get the PWM value directly from the hardware.

Read a raw value from a PWM channel.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel used for this Victor

Returns:

Raw PWM control value. Range: 0 - 255.

6.41.1.5 UINT8 GetVictorRaw (UINT32 *channel*)

Get the PWM value directly from the hardware.

Read a raw value from a PWM channel.

Parameters:

channel The PWM channel used for this Victor

Returns:

Raw PWM control value. Range: 0 - 255.

6.41.1.6 void SetVictorRaw (UINT32 *slot*, UINT32 *channel*, UINT8 *value*)

Set the PWM value directly to the hardware.

Write a raw value to a PWM channel.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel used for this Victor

value Raw PWM value. Range 0 - 255.

6.41.1.7 void SetVictorRaw (UINT32 *channel*, UINT8 *value*)

Set the PWM value directly to the hardware.

Write a raw value to a PWM channel.

Parameters:

channel The PWM channel used for this Victor

value Raw PWM value. Range 0 - 255.

6.41.1.8 void SetVictorSpeed (UINT32 *channel*, float *speed*)

Set the PWM value.

The PWM value is set using a range of -1.0 to 1.0, appropriately scaling the value for the FPGA.

Parameters:

channel The PWM channel used for this Victor

speed The speed value between -1.0 and 1.0 to set.

6.41.1.9 void SetVictorSpeed (UINT32 *slot*, UINT32 *channel*, float *speed*)

Set the PWM value.

The PWM value is set using a range of -1.0 to 1.0, appropriately scaling the value for the FPGA.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel used for this Victor

speed The speed value between -1.0 and 1.0 to set.

6.42 CVictor.h File Reference

```
#include <VxWorks.h>
```

Functions

- void SetVictorSpeed (UINT32 module, UINT32 channel, float speed)
- void SetVictorSpeed (UINT32 channel, float speed)
- void SetVictorRaw (UINT32 channel, UINT8 value)
- UINT8 GetVictorRaw (UINT32 channel)
- void SetVictorRaw (UINT32 module, UINT32 channel, UINT8 value)
- UINT8 GetVictorRaw (UINT32 module, UINT32 channel)
- void DeleteVictor (UINT32 module, UINT32 channel)
- void DeleteVictor (UINT32 channel)

6.42.1 Function Documentation

6.42.1.1 void DeleteVictor (UINT32 *channel*)

Delete resources for a Victor Free the underlying object and delete the allocated resources for the Victor

Parameters:

channel The PWM channel used for this Victor

6.42.1.2 void DeleteVictor (UINT32 *slot*, UINT32 *channel*)

Delete resources for a Victor Free the underlying object and delete the allocated resources for the Victor

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel used for this Victor

6.42.1.3 UINT8 GetVictorRaw (UINT32 *slot*, UINT32 *channel*)

Get the PWM value directly from the hardware.

Read a raw value from a PWM channel.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel used for this Victor

Returns:

Raw PWM control value. Range: 0 - 255.

6.42.1.4 UINT8 GetVictorRaw (UINT32 *channel*)

Get the PWM value directly from the hardware.

Read a raw value from a PWM channel.

Parameters:

channel The PWM channel used for this Victor

Returns:

Raw PWM control value. Range: 0 - 255.

6.42.1.5 void SetVictorRaw (UINT32 *slot*, UINT32 *channel*, UINT8 *value*)

Set the PWM value directly to the hardware.

Write a raw value to a PWM channel.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel used for this Victor

value Raw PWM value. Range 0 - 255.

6.42.1.6 void SetVictorRaw (UINT32 *channel*, UINT8 *value*)

Set the PWM value directly to the hardware.

Write a raw value to a PWM channel.

Parameters:

channel The PWM channel used for this Victor

value Raw PWM value. Range 0 - 255.

6.42.1.7 void SetVictorSpeed (UINT32 *channel*, float *speed*)

Set the PWM value.

The PWM value is set using a range of -1.0 to 1.0, appropriately scaling the value for the FPGA.

Parameters:

channel The PWM channel used for this Victor

speed The speed value between -1.0 and 1.0 to set.

6.42.1.8 void SetVictorSpeed (UINT32 *slot*, UINT32 *channel*, float *speed*)

Set the PWM value.

The PWM value is set using a range of -1.0 to 1.0, appropriately scaling the value for the FPGA.

Parameters:

slot The slot the digital module is plugged into

channel The PWM channel used for this Victor

speed The speed value between -1.0 and 1.0 to set.

6.43 CWrappers.h File Reference

Typedefs

- typedef SensorBase *(* SensorCreator)(UINT32 slot, UINT32 channel)

6.43.1 Typedef Documentation

6.43.1.1 typedef SensorBase*(* SensorCreator)(UINT32 slot, UINT32 channel)

6.44 SimpleCRobot.cpp File Reference

```
#include "SimpleCRobot.h"
#include "Timer.h"
#include "Utility.h"
```

Functions

- bool IsAutonomous ()
- bool IsOperatorControl ()
- bool IsDisabled ()
- void SetWatchdogEnabled (bool enable)
- void SetWatchdogExpiration (float time)
- void WatchdogFeed ()

Variables

- static SimpleCRobot * simpleCRobot = NULL

6.44.1 Function Documentation

6.44.1.1 bool IsAutonomous ()

Returns flag for field state

Returns:

true if the field is in Autonomous mode

6.44.1.2 bool IsDisabled ()

Returns the robot state

Returns:

true if the robot is disabled

6.44.1.3 bool IsOperatorControl ()

Returns flag for field state

Returns:

true if the field is in Operator Control mode (teleop)

6.44.1.4 void SetWatchdogEnabled (bool *enable*)

6.44.1.5 void SetWatchdogExpiration (float *time*)

6.44.1.6 void WatchdogFeed ()

6.44.2 Variable Documentation

6.44.2.1 SimpleCRobot* simpleCRobot = NULL [static]

6.45 SimpleCRobot.h File Reference

```
#include "RobotBase.h"
```

Data Structures

- class SimpleCRobot

Functions

- void Autonomous ()
- void OperatorControl ()
- void Initialize ()
- bool IsAutonomous ()
- bool IsOperatorControl ()
- bool IsDisabled ()
- void SetWatchdogEnabled (bool enable)
- void SetWatchdogExpiration (float time)
- void WatchdogFeed ()

6.45.1 Function Documentation

6.45.1.1 void Autonomous ()

6.45.1.2 void Initialize ()

6.45.1.3 bool IsAutonomous ()

Returns flag for field state

Returns:

true if the field is in Autonomous mode

6.45.1.4 bool IsDisabled ()

Returns the robot state

Returns:

true if the robot is disabled

6.45.1.5 bool IsOperatorControl ()

Returns flag for field state

Returns:

true if the field is in Operator Control mode (teleop)

6.45.1.6 void OperatorControl ()

6.45.1.7 void SetWatchdogEnabled (bool *enable*)

6.45.1.8 void SetWatchdogExpiration (float *time*)

6.45.1.9 void WatchdogFeed ()