

[Todo List](#)

[Deprecated List](#)

[Bug List](#)

[Class List](#)

[AbstractNetworkTableEntryStore](#)

[Accelerometer](#)

[ADXL345\\_I2C](#)

[ADXL345\\_SPI](#)

[ADXL345\\_SPI::AllAxes](#)

[ADXL345\\_I2C::AllAxes](#)

[AnalogChannel](#)

[AnalogIOButton](#)

[AnalogModule](#)

[AnalogTrigger](#)

[AnalogTriggerOutput](#)

[ArrayData](#)

[ArrayEntryData](#)

[ArrayEntryType](#)

[AxisCamera](#)

[AxisCameraParams](#)

[BadMessageException](#)

[BinaryImage](#)

[BooleanArray](#)

[Button](#)

[ButtonScheduler](#)

[CANJaguar](#)

[ClientConnectionAdapter](#)

[ClientConnectionListenerManager](#)

[ClientConnectionState](#)

[ClientConnectionState\\_Error](#)

[ClientConnectionState](#) [ProtocolUnsupportedByServer](#)  
[ClientNetworkTableEntryStore](#)  
[ColorImage](#)  
[ColorReport\\_struct](#)  
[Command](#)  
[CommandGroup](#)  
[CommandGroupEntry](#)  
[ComplexData](#)  
[ComplexEntryType](#)  
[Compressor](#)  
[ConnectionAdapter](#)  
[ConnectionMonitorThread](#)  
[Controller](#)  
[Counter](#)  
[CounterBase](#)  
[Dashboard](#)  
[DashboardBase](#)  
[DataIOStream](#)  
[DefaultEntryTypes](#)  
[DefaultThreadManager](#)  
[DigitalInput](#)  
[DigitalIOButton](#)  
[DigitalModule](#)  
[DigitalOutput](#)  
[DigitalSource](#)  
[DoubleSolenoid](#)  
[DriverStation](#)  
[DriverStationEnhancedIO](#)  
[DriverStationLCD](#)  
[Encoder](#)  
[EntryCache](#)

[EntryValue](#)  
[EnumCameraParameter](#)  
[EOFException](#)  
[Error](#)  
[ErrorBase](#)  
[FDIOTStream](#)  
[FlushableOutgoingEntryReceiver](#)  
[GearTooth](#)  
[GenericHID](#)  
[Gyro](#)  
[HeldButtonScheduler](#)  
[HiTechnicColorSensor](#)  
[HiTechnicCompass](#)  
[HSLImage](#)  
[I2C](#)  
[IllegalStateException](#)  
[ImageBase](#)  
[IncomingEntryReceiver](#)  
[IntCameraParameter](#)  
[InternalButton](#)  
[InterruptableSensorBase](#)  
[IOException](#)  
[IOStream](#)  
[IOStreamFactory](#)  
[IOStreamProvider](#)  
[IRemote](#)  
[IRemoteConnectionListener](#)  
[ITable](#)  
[ITableListener](#)  
[ITableProvider](#)  
[IterativeRobot](#)

[Jaguar](#)  
[Skeleton::Joint](#)  
[Joystick](#)  
[JoystickButton](#)  
[Kinect](#)  
[KinectStick](#)  
[LiveWindow](#)  
[LiveWindowSendable](#)  
[LiveWindowStatusListener](#)  
[Module](#)  
[MonoImage](#)  
[MotorSafety](#)  
[MotorSafetyHelper](#)  
[NamedSendable](#)  
[NetworkButton](#)  
[NetworkTable](#)  
[NetworkTableClient](#)  
[NetworkTableClientMode](#)  
[NetworkTableConnection](#)  
[NetworkTableConnectionListenerAdapter](#)  
[NetworkTableEntry](#)  
[NetworkTableEntryType](#)  
[NetworkTableEntryTypeManager](#)  
[NetworkTableKeyCache](#)  
[NetworkTableKeyListenerAdapter](#)  
[NetworkTableListenerAdapter](#)  
[NetworkTableMode](#)  
[NetworkTableName](#)  
[NetworkTableProvider](#)  
[NetworkTableServer](#)  
[NetworkTableServerMode](#)

[NetworkTableSubListenerAdapter](#)  
[Notifier](#)  
[NTThread](#)  
[NTThreadManager](#)  
[NumberArray](#)  
[OutgoingEntryReceiver](#)  
[OutgoingEntryReceiver\\_NULL\\_t](#)  
[ParticleAnalysisReport\\_struct](#)  
[pcre\\_callout\\_block](#)  
[pcre\\_extra](#)  
[PCVideoServer](#)  
[PeriodicNTThread](#)  
[PeriodicRunnable](#)  
[PIDCommand](#)  
[PIDController](#)  
[PIDOutput](#)  
[PIDSource](#)  
[PIDSubsystem](#)  
[Kinect::Point4](#)  
[Preferences](#)  
[PressedButtonScheduler](#)  
[PrintCommand](#)  
[PWM](#)  
[ReentrantSemaphore](#)  
[Relay](#)  
[ReleasedButtonScheduler](#)  
[Resource](#)  
[HiTechnicColorSensor::RGB](#)  
[RGBImage](#)  
[RobotBase](#)  
[RobotDeleter](#)

[RobotDrive](#)  
[SafePWM](#)  
[Scheduler](#)  
[ScopedSocket](#)  
[Sendable](#)  
[SendableChooser](#)  
[SensorBase](#)  
[SerialPort](#)  
[ServerAdapterManager](#)  
[ServerConnectionAdapter](#)  
[ServerConnectionList](#)  
[ServerConnectionState](#)  
[ServerConnectionState\\_Error](#)  
[ServerIncomingConnectionListener](#)  
[ServerIncomingStreamMonitor](#)  
[ServerNetworkTableEntryStore](#)  
[Servo](#)  
[SimpleRobot](#)  
[Skeleton](#)  
[SmartDashboard](#)  
[SocketServerStreamProvider](#)  
[SocketStreamFactory](#)  
[SocketStreams](#)  
[Solenoid](#)  
[SolenoidBase](#)  
[SpeedController](#)  
[SPI](#)  
[StartCommand](#)  
[StringArray](#)  
[StringCache](#)  
[Subsystem](#)

[Synchronized](#)

[TableKeyExistsWithDifferentTypeException](#)

[TableKeyNotDefinedException](#)

[TableListenerManager](#)

[Talon](#)

[Task](#)

[Threshold](#)

[Timer](#)

[TrackingThreshold\\_struct](#)

[TransactionDirtier](#)

[Trigger](#)

[Ultrasonic](#)

[Victor](#)

[WaitCommand](#)

[WaitForChildren](#)

[WaitUntilCommand](#)

[Watchdog](#)

[WriteManager](#)

[Class Hierarchy](#)

[ADXL345\\_SPI::AllAxes](#)

[ADXL345\\_I2C::AllAxes](#)

[ArrayEntryData](#)

[BadMessageException](#)

[ButtonScheduler](#)

[HeldButtonScheduler](#)

[PressedButtonScheduler](#)

[ReleasedButtonScheduler](#)

[ClientConnectionListenerManager](#)

[NetworkTableNode](#)

[NetworkTableClient](#)

[NetworkTableServer](#)

[ClientConnectionState](#)  
[ClientConnectionState\\_Error](#)  
[ClientConnectionState\\_ProtocolUnsupportedByServer](#)  
[ColorReport\\_struct](#)  
[CommandGroupEntry](#)  
[ComplexData](#)  
[ArrayData](#)  
[BooleanArray](#)  
[NumberArray](#)  
[StringArray](#)  
[ConnectionAdapter](#)  
[ClientConnectionAdapter](#)  
[ServerConnectionAdapter](#)  
[Controller](#)  
[PIDController](#)  
[CounterBase](#)  
[Counter](#)  
[GearTooth](#)  
[Encoder](#)  
[DataIOStream](#)  
[DefaultEntryTypes](#)  
[EntryCache](#)  
[EntryValue](#)  
[Error](#)  
[ErrorBase](#)  
[AxisCameraParams](#)  
[AxisCamera](#)  
[CANJaguar](#)  
[Command](#)  
[CommandGroup](#)  
[PIDCommand](#)

[PrintCommand](#)  
[StartCommand](#)  
[WaitCommand](#)  
[WaitForChildren](#)  
[WaitUntilCommand](#)  
[DashboardBase](#)  
[Dashboard](#)  
[DriverStationEnhancedIO](#)  
[ImageBase](#)  
[ColorImage](#)  
[HSImage](#)  
[RGBImage](#)  
[MonoImage](#)  
[BinaryImage](#)  
[Joystick](#)  
[KinectStick](#)  
[MotorSafetyHelper](#)  
[Notifier](#)  
[PCVideoServer](#)  
[Preferences](#)  
[Resource](#)  
[RobotDrive](#)  
[Scheduler](#)  
[SensorBase](#)  
[Accelerometer](#)  
[ADXL345\\_I2C](#)  
[ADXL345\\_SPI](#)  
[AnalogChannel](#)  
[AnalogTrigger](#)  
[Compressor](#)  
[Counter](#)

[DriverStation](#)  
[DriverStationLCD](#)  
[Encoder](#)  
[Gyro](#)  
[HiTechnicColorSensor](#)  
[HiTechnicCompass](#)  
[I2C](#)  
[InterruptableSensorBase](#)  
[DigitalSource](#)  
[AnalogTriggerOutput](#)  
[DigitalInput](#)  
[DigitalOutput](#)  
[Kinect](#)  
[Module](#)  
[AnalogModule](#)  
[DigitalModule](#)  
[PWM](#)  
[SafePWM](#)  
[Jaguar](#)  
[Servo](#)  
[Talon](#)  
[Victor](#)  
[Relay](#)  
[SmartDashboard](#)  
[SolenoidBase](#)  
[DoubleSolenoid](#)  
[Solenoid](#)  
[SPI](#)  
[Ultrasonic](#)  
[Watchdog](#)  
[SerialPort](#)

[Subsystem](#)

[PIDSubsystem](#)

[Task](#)

[GenericHID](#)

[Joystick](#)

[KinectStick](#)

[IllegalStateException](#)

[IncomingEntryReceiver](#)

[AbstractNetworkTableEntryStore](#)

[ClientNetworkTableEntryStore](#)

[ServerNetworkTableEntryStore](#)

[ClientConnectionAdapter](#)

[ServerConnectionAdapter](#)

[IntCameraParameter](#)

[EnumCameraParameter](#)

[IOException](#)

[EOFException](#)

[IOStream](#)

[FDIOTStream](#)

[IOStreamFactory](#)

[SocketStreamFactory](#)

[IOStreamProvider](#)

[SocketServerStreamProvider](#)

[IRemote](#)

[NetworkTable](#)

[NetworkTableNode](#)

[IRemoteConnectionListener](#)

[NetworkTableConnectionListenerAdapter](#)

[ITable](#)

[NetworkTable](#)

[ITableListener](#)

[CANJaguar](#)  
[Command](#)  
[DigitalOutput](#)  
[DoubleSolenoid](#)  
[LiveWindowStatusListener](#)  
[NetworkTableKeyListenerAdapter](#)  
[NetworkTableListenerAdapter](#)  
[NetworkTableSubListenerAdapter](#)  
[PIDController](#)  
[Preferences](#)  
[PWM](#)  
[Relay](#)  
[Solenoid](#)  
[ITableProvider](#)  
[NetworkTableProvider](#)  
[Skeleton::Joint](#)  
[LiveWindow](#)  
[MotorSafety](#)  
[CANJaguar](#)  
[RobotDrive](#)  
[SafePWM](#)  
[NetworkTableConnection](#)  
[NetworkTableEntry](#)  
[NetworkTableEntryType](#)  
[ComplexEntryType](#)  
[ArrayEntryType](#)  
[NetworkTableEntryTypeManager](#)  
[NetworkTableMode](#)  
[NetworkTableClientMode](#)  
[NetworkTableServerMode](#)  
[NTThread](#)

[PeriodicNTThread](#)  
[NTThreadManager](#)  
[DefaultThreadManager](#)  
[OutgoingEntryReceiver](#)  
[FlushableOutgoingEntryReceiver](#)  
[ClientConnectionAdapter](#)  
[ServerConnectionAdapter](#)  
[ServerConnectionList](#)  
[OutgoingEntryReceiver\\_NULL\\_t](#)  
[TransactionDirtier](#)  
[WriteManager](#)  
[ParticleAnalysisReport\\_struct](#)  
[pcre\\_callout\\_block](#)  
[pcre\\_extra](#)  
[PeriodicRunnable](#)  
[ConnectionMonitorThread](#)  
[ServerIncomingStreamMonitor](#)  
[WriteManager](#)  
[PIDOutput](#)  
[PIDCommand](#)  
[PIDS subsystem](#)  
[SpeedController](#)  
[CANJaguar](#)  
[Jaguar](#)  
[Talon](#)  
[Victor](#)  
[PIDS source](#)  
[Accelerometer](#)  
[AnalogChannel](#)  
[Encoder](#)  
[Gyro](#)

[PIDCommand](#)  
[PIDSubsystem](#)  
[Ultrasonic](#)  
[Kinect::Point4](#)  
[ReentrantSemaphore](#)  
[HiTechnicColorSensor::RGB](#)  
[RobotBase](#)  
[IterativeRobot](#)  
[SimpleRobot](#)  
[RobotDeleter](#)  
[ScopedSocket](#)  
[Sendable](#)  
[LiveWindowSendable](#)  
[Accelerometer](#)  
[AnalogChannel](#)  
[CANJaguar](#)  
[Compressor](#)  
[Counter](#)  
[DigitalInput](#)  
[DigitalOutput](#)  
[DoubleSolenoid](#)  
[Encoder](#)  
[Gyro](#)  
[HiTechnicCompass](#)  
[PIDController](#)  
[PWM](#)  
[Relay](#)  
[Solenoid](#)  
[Ultrasonic](#)  
[NamedSendable](#)  
[Command](#)

[Subsystem](#)[SendableChooser](#)[Trigger](#)[AnalogIOButton](#)[Button](#)[DigitalIOButton](#)[InternalButton](#)[JoystickButton](#)[NetworkButton](#)[ServerAdapterManager](#)[ServerConnectionList](#)[ServerConnectionState](#)[ServerConnectionState\\_Error](#)[ServerIncomingConnectionListener](#)[NetworkTableServer](#)[Skeleton](#)[SocketStreams](#)[StringCache](#)[NetworkTableKeyCache](#)[Synchronized](#)[TableKeyExistsWithDifferentTypeException](#)[TableKeyNotDefinedException](#)[TableListenerManager](#)[NetworkTableName](#)[Threshold](#)[Timer](#)[TrackingThreshold\\_struct](#)[Class Members](#)[Graphical Class Hierarchy](#)[File List](#)



# **Todo List**

---

## **Member `AnalogModule::VoltsToValue (INT32 channel, float voltage)`**

This assumes raw values. Oversampling not supported as is.

## **Class `HiTechnicCompass`**

Implement a calibration method for the sensor.

## **Class `SimpleRobot`**

If this is going to last until release, it needs a better name.



# Deprecated List

---

## Member **CANJaguar::Disable ()**

Call DisableControl instead.

## Member **CANJaguar::PIDWrite (float output)**

Call Set instead.

## Member **CANJaguar::StopMotor ()**

Call DisableControl instead.

## Member **Encoder::GetPeriod ()**

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

## Member **Encoder::SetMaxPeriod (double maxPeriod)**

Use SetMinRate() in favor of this method. This takes unscaled periods and SetMinRate() scales using value from SetDistancePerPulse().



# Bug List

---

## **Member `SerialPort::Printf (const char *writeFmt,...)`**

All pointer-based parameters seem to return an error.

## **Member `SerialPort::Scanf (const char *readFmt,...)`**

All pointer-based parameters seem to return an error.



# Class List

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

<a href="#">AbstractNetworkTableEntryStore</a>	
<a href="#">Accelerometer</a>	Handle operation of the <a href="#">Accelerometer</a>
<a href="#">ADXL345_I2C</a>	ADXL345 <a href="#">Accelerometer</a> over I2C
<a href="#">ADXL345_SPI</a>	ADXL345 <a href="#">Accelerometer</a> over SPI
<a href="#">ADXL345_SPI::AllAxes</a>	
<a href="#">ADXL345_I2C::AllAxes</a>	
<a href="#">AnalogChannel</a>	Analog channel class
<a href="#">AnalogIOButton</a>	
<a href="#">AnalogModule</a>	Analog <a href="#">Module</a> class
<a href="#">AnalogTrigger</a>	
<a href="#">AnalogTriggerOutput</a>	Class to represent a specific output from an analog trigger
<a href="#">ArrayData</a>	
<a href="#">ArrayEntryData</a>	
<a href="#">ArrayEntryType</a>	
<a href="#">AxisCamera</a>	<a href="#">AxisCamera</a> class
<a href="#">AxisCameraParams</a>	<a href="#">AxisCameraParams</a> class
<a href="#">BadMessageException</a>	
<a href="#">BinaryImage</a>	
<a href="#">BooleanArray</a>	
<a href="#">Button</a>	This class provides an easy way to send commands to OI inputs
<a href="#">ButtonScheduler</a>	
<a href="#">CANJaguar</a>	Luminary Micro <a href="#">Jaguar</a> S CAN interface
<a href="#">ClientConnectionAdapter</a>	Object that adapts message formats between client and server
<a href="#">ClientConnectionListenerManager</a>	An object that manages connections to clients, handles listeners and fires events to them
<a href="#">ClientConnectionState</a>	Represents a state that the connection is in
	Represents that the client has disconnected

<b>ClientConnectionState_Error</b>	state
<b>ClientConnectionState_ProtocolUnsupportedByServer</b>	Represents that a client received a message from the server that the client's protocol is not supported by the server
<b>ClientNetworkTableEntryStore</b>	The entry store for a <b>NetworkTableClient</b>
<b>ColorImage</b>	
<b>ColorReport_struct</b>	Tracking functions return this structure
<b>Command</b>	At the very core of the entire command framework
<b>CommandGroup</b>	A <b>CommandGroup</b> is a group of commands which are executed in sequence
<b>CommandGroupEntry</b>	
<b>ComplexData</b>	
<b>ComplexEntryType</b>	
<b>Compressor</b>	<b>Compressor</b> object
<b>ConnectionAdapter</b>	
<b>ConnectionMonitorThread</b>	A periodic thread that represents a connection
<b>Controller</b>	Interface for Controllers (the interface for controllers)
<b>Counter</b>	Class for counting the number of ticks on a digital input channel
<b>CounterBase</b>	Interface for counting the number of ticks on a digital input channel
<b>Dashboard</b>	Pack data into the "user object" that gets sent to the dashboard via the driver station
<b>DashboardBase</b>	
<b>DataIOStream</b>	
<b>DefaultEntryTypes</b>	
<b>DefaultThreadManager</b>	
<b>DigitalInput</b>	Class to read a digital input
<b>DigitalIOButton</b>	

DigitalModule	
DigitalOutput	Class to write to digital output
DigitalSource	<b>DigitalSource</b> Interface
DoubleSolenoid	<b>DoubleSolenoid</b> class for channels of high voltage (9472 module)
DriverStation	Provide access to the network communication data to / from Driver Station
DriverStationEnhancedIO	Interact with the more complete available from the newer station
DriverStationLCD	Provide access to "LCD" on Driver Station
Encoder	Class to read quad encoder
EntryCache	
EntryValue	
EnumCameraParameter	Enumerated camera parameter
EOFException	
Error	<b>Error</b> object represents an error condition
ErrorBase	Base class for most objects
FDIOutputStream	
FlushableOutgoingEntryReceiver	
GearTooth	Alias for counter class
GenericHID	<b>GenericHID</b> Interface
Gyro	Use a rate gyro to return heading relative to a starting point
HeldButtonScheduler	
HiTechnicColorSensor	HiTechnic NXT Color Sensor
HiTechnicCompass	HiTechnic NXT Compass
HSLImage	A color image represented in the sRGB color space at 3 bytes per pixel
I2C	<b>I2C</b> bus interface class
IllegalStateException	
ImageBase	

<b>IncomingEntryReceiver</b>	
<b>IntCameraParameter</b>	Integer camera parameter
<b>InternalButton</b>	
<b>InterruptableSensorBase</b>	
<b>IOException</b>	
<b>IOStream</b>	
<b>IOStreamFactory</b>	A factory that will create <b>IOStream</b>
<b>IOStreamProvider</b>	An object that will provide <b>IOStream</b> of clients to a <b>NetworkTable</b> Server
<b>IRemote</b>	Represents an object that handles a remote connection
<b>IRemoteConnectionListener</b>	A listener that listens for changes in a <b>IRemote</b> object
<b>ITable</b>	
<b>ITableListener</b>	A listener that listens to changes in values in a <b>ITable</b>
<b>ITableProvider</b>	A simple interface to provide data to a <b>ITable</b>
<b>IterativeRobot</b>	<b>IterativeRobot</b> implements the type of Robot Program framework extending the <b>RobotBase</b>
<b>Jaguar</b>	Luminary Micro <b>Jaguar</b> Servo
<b>Skeleton::Joint</b>	
<b>Joystick</b>	Handle input from standard joysticks connected to the Driver Station
<b>JoystickButton</b>	
<b>Kinect</b>	Handles raw data input from the <b>Kinect</b> Server when used with a <b>Kinect</b> device connected to the Driver Station
<b>KinectStick</b>	Handles input from the <b>J</b> oystick sent by the FRC <b>Kinect</b> Stick used with a <b>Kinect</b> device connected to the Driver Station
	Public interface for putting data into the NetworkTables

<b>LiveWindow</b>	actuators on the <b>LiveWindow</b>
<b>LiveWindowSendable</b>	Live Window <b>Sendable</b> is a type of object sendable to a live window
<b>LiveWindowStatusListener</b>	
<b>Module</b>	
<b>MonoImage</b>	
<b>MotorSafety</b>	
<b>MotorSafetyHelper</b>	
<b>NamedSendable</b>	The interface for sendables that gives the sendable a default name for the Smart <b>Dashboard</b>
<b>NetworkButton</b>	
<b>NetworkTable</b>	
<b>NetworkTableClient</b>	A client node in NetworkTables
<b>NetworkTableClientMode</b>	
<b>NetworkTableConnection</b>	
<b>NetworkTableConnectionListenerAdapter</b>	
<b>NetworkTableEntry</b>	An entry in a network table
<b>NetworkTableEntryType</b>	
<b>NetworkTableEntryTypeManager</b>	
<b>NetworkTableKeyCache</b>	
<b>NetworkTableKeyListenerAdapter</b>	
<b>NetworkTableListenerAdapter</b>	
<b>NetworkTableMode</b>	Represents a different mode that network tables can be configured in
<b>NetworkTreeNode</b>	Node (either a client or a server) in network tables 2.0. Implementers of the class must call <b>init(NetworkTableTransport)</b> and <b>AbstractNetworkTable</b> before calling any other methods in this class
<b>NetworkTableProvider</b>	

<b>NetworkTableServer</b>	A server node in Network
<b>NetworkTableServerMode</b>	
<b>NetworkTableSubListenerAdapter</b>	
<b>Notifier</b>	
<b>NTThread</b>	Represents a thread in the tables system
<b>NTThreadManager</b>	A thread manager that can obtain new threads
<b>NumberArray</b>	
<b>OutgoingEntryReceiver</b>	
<b>OutgoingEntryReceiver_NULL_t</b>	
<b>ParticleAnalysisReport_struct</b>	FrcParticleAnalysis return structure
<b>pcre_callout_block</b>	
<b>pcre_extra</b>	
<b>PCVideoServer</b>	Class that serves images to clients
<b>PeriodicNTThread</b>	
<b>PeriodicRunnable</b>	A runnable where the run() method will be called periodically
<b>PIDCommand</b>	
<b>PIDController</b>	Class implements a PID controller
<b>PIDOutput</b>	<b>PIDOutput</b> interface is a virtual base class for output for the PID class
<b>PIDSource</b>	<b>PIDSource</b> interface is a virtual base class for sensor source for the PID class
<b>PIDSubsystem</b>	This class is designed to handle the case where there is a <b>Servo</b> which uses a single <b>PIDController</b> almost constantly (for instance an elevator which attempts to maintain constant height)
<b>Kinect::Point4</b>	
<b>Preferences</b>	The preferences class provides a relatively simple way to save important values to the configuration file so they can be loaded the next time the cRIO is powered up

<b>PressedButtonScheduler</b>	
<b>PrintCommand</b>	
<b>PWM</b>	Class implements the <a href="#">PWM</a> generation in the FPGA
<b>ReentrantSemaphore</b>	Wrap a vxWorks semaphore for easier use in C++
<b>Relay</b>	Class for Spike style relay
<b>ReleasedButtonScheduler</b>	
<b>Resource</b>	Convenient way to track resources
<b>HiTechnicColorSensor::RGB</b>	
<b>RGBImage</b>	A color image represented in space at 3 bytes per pixel
<b>RobotBase</b>	Implement a Robot Program framework
<b>RobotDeleter</b>	This class exists for the sake of getting its destructor called when the module unloads
<b>RobotDrive</b>	Utility class for handling I/O based on a definition of the configuration
<b>SafePWM</b>	A safe version of the <a href="#">PWM</a>
<b>Scheduler</b>	
<b>ScopedSocket</b>	Implements an object that automatically does a close of the camera socket on destruction
<b>Sendable</b>	
<b>SendableChooser</b>	The <b>SendableChooser</b> is a useful tool for presenting options to the <a href="#">SmartDashboard</a>
<b>SensorBase</b>	Base class for all sensors
<b>SerialPort</b>	Driver for the RS-232 serial port on cRIO
<b>ServerAdapterManager</b>	A class that manages connections to the server
<b>ServerConnectionAdapter</b>	Object that adapts message from client to the server

<b>ServerConnectionList</b>	A list of connections that currently has
<b>ServerConnectionState</b>	Represents the state of a connection to the server
<b>ServerConnectionState_Error</b>	Represents that the client has an error state
<b>ServerIncomingConnectionListener</b>	Listener for new incoming connections
<b>ServerIncomingStreamMonitor</b>	Thread that monitors for incoming connections
<b>ServerNetworkTableEntryStore</b>	The entry store for a <b>NetworkTableServer</b>
<b>Servo</b>	Standard hobby style servos
<b>SimpleRobot</b>	
<b>Skeleton</b>	Represents <b>Skeleton data</b> from a <b>Kinect</b> device connected to a <b>Kinect Station</b>
<b>SmartDashboard</b>	
<b>SocketServerStreamProvider</b>	
<b>SocketStreamFactory</b>	
<b>SocketStreams</b>	Static factory for socket streams factories and providers
<b>Solenoid</b>	<b>Solenoid</b> class for running DC voltage Digital Output (9V)
<b>SolenoidBase</b>	<b>SolenoidBase</b> class is the base class for the <b>Solenoid</b> and <b>DoubleSolenoid</b> classes
<b>SpeedController</b>	Interface for speed controllers
<b>SPI</b>	<b>SPI</b> bus interface class
<b>StartCommand</b>	
<b>StringArray</b>	
<b>StringCache</b>	A simple cache that allows the mapping of one string to a calculated one
<b>Subsystem</b>	Provide easy support for

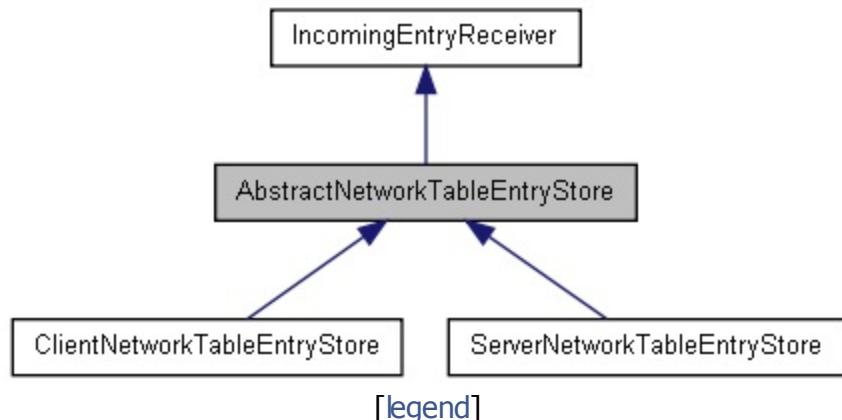
<b>Synchronized</b>	regions
<b>TableKeyExistsWithDifferentTypeException</b>	Throw to indicate that an attempt to put data to a table is illegal because the specified key exists with a different data type than the put data.
<b>TableKeyNotDefinedException</b>	An exception throw when a key-value fails in a <b>ITable</b> .
<b>TableListenerManager</b>	
<b>Talon</b>	CTRE <b>Talon</b> Speed Controller
<b>Task</b>	WPI task is a wrapper for a <b>Task</b> object
<b>Threshold</b>	Color threshold values
<b>Timer</b>	<b>Timer</b> objects measure a time interval in seconds
<b>TrackingThreshold_struct</b>	
<b>TransactionDirtier</b>	A transaction receiver that marks Table entries as dirty in the database.
<b>Trigger</b>	This class provides an easy way to map commands to inputs.
<b>Ultrasonic</b>	<b>Ultrasonic</b> rangefinder component
<b>Victor</b>	IFI <b>Victor</b> Speed Controller
<b>WaitCommand</b>	
<b>WaitForChildren</b>	
<b>WaitUntilCommand</b>	
<b>Watchdog</b>	<b>Watchdog</b> timer class
<b>WriteManager</b>	A write manager is a <b>IncomingEntryReceiver</b> that receives transactions and then analyzes them to determine if they can be flushed. It dispatches them to a flusher which then sends them to a transaction receiver that offered all queued transactions and then flushed.

[Class List](#)[Class Hierarchy](#)[Class Members](#)

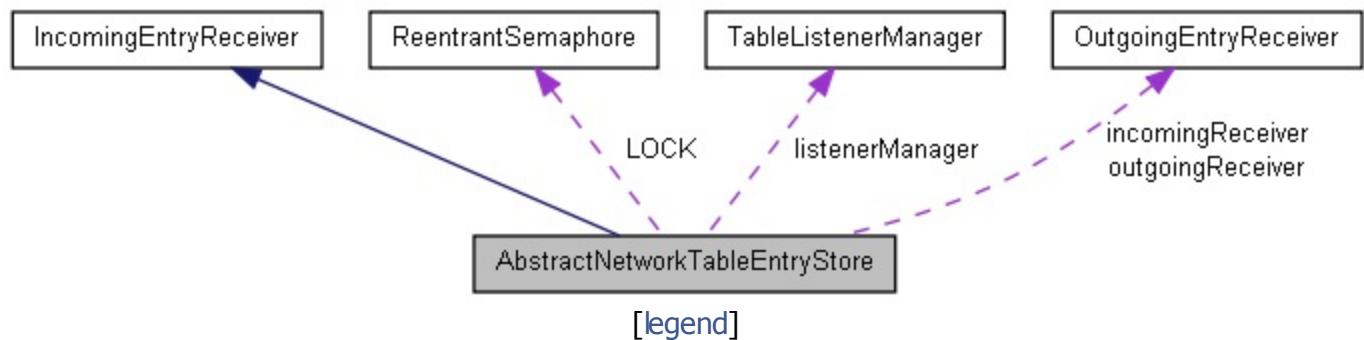
[Public Member Functions](#) | [Public Attributes](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#)

# AbstractNetworkTableEntryStore Class Reference

Inheritance diagram for AbstractNetworkTableEntryStore:



Collaboration diagram for AbstractNetworkTableEntryStore:



List of all members.

# Public Member Functions

**NetworkTableEntry \*** **GetEntry** (EntryId entryId)

**NetworkTableEntry \*** **GetEntry** (std::string &name)  
Get an entry based on it's name.

**std::vector< std::string > \*** **keys** ()

void **clearEntries** ()

Remove all entries NOTE: This method should not be used with applications which cache entries which would lead to unknown results This method is for use in testing only.

void **clearIds** ()

clear the id's of all entries

void **SetOutgoingReceiver** (**OutgoingEntryReceiver** \*receiver)

void **SetIncomingReceiver** (**OutgoingEntryReceiver** \*receiver)

void **PutOutgoing** (std::string &name,

**NetworkTableEntryType** \*type, **EntryValue** value)

Stores the given value under the given name and queues it for transmission to the server.

void **PutOutgoing** (**NetworkTableEntry** \*tableEntry,  
**EntryValue** value)

void **offerIncomingAssignment** (**NetworkTableEntry** \*entry)

void **offerIncomingUpdate** (**NetworkTableEntry** \*entry,  
EntryId sequenceNumber, **EntryValue** value)

void **notifyEntries** (**ITable** \*table, **ITableListener** \*listener)

Called to say that a listener should notify the listener manager of all of the entries.

## Public Attributes

**ReentrantSemaphore LOCK**

# Protected Member Functions

**AbstractNetworkTableEntryStore**

(**TableListenerManager** &lstrnManager)

virtual bool **addEntry** (**NetworkTableEntry** \*entry)=0

virtual bool **updateEntry** (**NetworkTableEntry** \*entry,  
SequenceNumber sequenceNumber, **EntryValue**  
value)=0

std::map< **EntryId**,

**NetworkTableEntry** \* > **idEntries**

std::map< std::string,

**NetworkTableEntry** \* > **namedEntries**

**TableListenerManager** & **listenerManager**

**OutgoingEntryReceiver** \* **outgoingReceiver**

**OutgoingEntryReceiver** \* **incomingReceiver**

# Member Function Documentation

## **NetworkTableEntry \* AbstractNetworkTableEntryStore::GetEntry ( std::string**

Get an entry based on it's name.

### **Parameters:**

**name** the name of the entry to look for

### **Returns:**

the entry or null if the entry does not exist

## **void AbstractNetworkTableEntryStore::notifyEntries ( ITable \* table, ITableListener \* listener )**

Called to say that a listener should notify the listener manager of all of the entries.

### **Parameters:**

**listener**  
**table**

## **void AbstractNetworkTableEntryStore::PutOutgoing ( std::string & NetworkTableEntryType EntryValue )**

Stores the given value under the given name and queues it for transmission to the server.

### **Parameters:**

**name** The name under which to store the given value.  
**type** The type of the given value.  
**value** The value to store.

### **Exceptions:**

**TableKeyExistsWithDifferentTypeException** Thrown if an entry already exists with the given name and is of a different type.

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

- C:/WindRiver/workspace/WPIlib/networktables2/**AbstractNetworkTableEntryStore.h**
- C:/WindRiver/workspace/WPIlib/networktables2/AbstractNetworkTableEntryStore.cpp

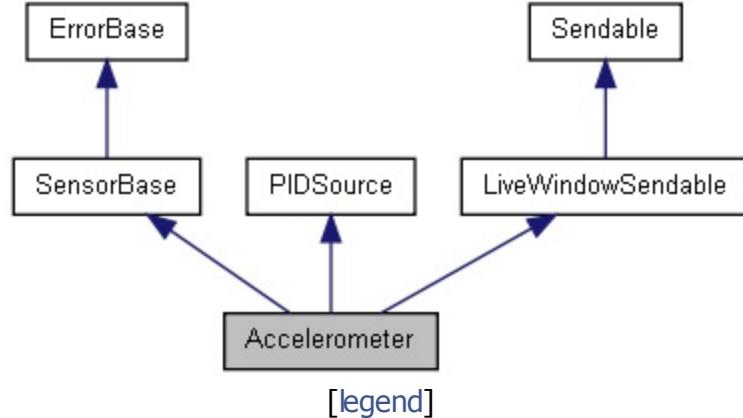


# Accelerometer Class Reference

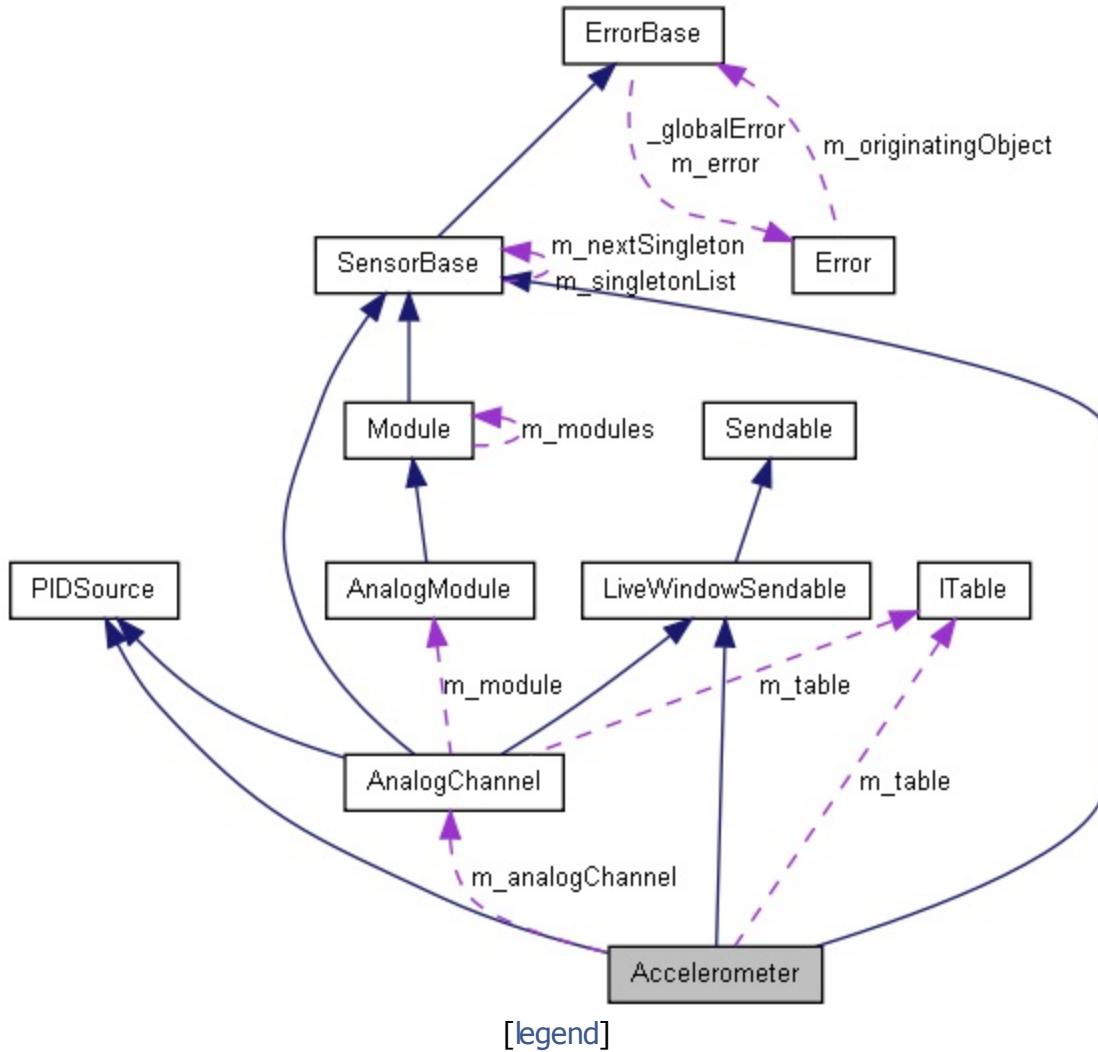
Handle operation of the accelerometer. [More...](#)

```
#include <Accelerometer.h>
```

Inheritance diagram for Accelerometer:



Collaboration diagram for Accelerometer:



List of all members.

# Public Member Functions

## **Accelerometer** (UINT32 channel)

Create a new instance of an accelerometer.

## **Accelerometer** (UINT8 moduleNumber, UINT32 channel)

Create new instance of accelerometer.

## **Accelerometer** (**AnalogChannel** \*channel)

Create a new instance of **Accelerometer** from an existing

## **AnalogChannel**.

**virtual ~Accelerometer ()**

Delete the analog components used for the accelerometer.

**float GetAcceleration ()**

Return the acceleration in Gs.

**void SetSensitivity (float sensitivity)**

Set the accelerometer sensitivity.

**void SetZero (float zero)**

Set the voltage that corresponds to 0 G.

**double PIDGet ()**

Get the Acceleration for the PID Source parent.

**void UpdateTable ()**

Update the table for this sendable object with the latest values.

**void StartLiveWindowMode ()**

Start having this sendable object automatically respond to value changes reflect the value on the table.

**void StopLiveWindowMode ()**

Stop having this sendable object automatically respond to value changes.

**std::string GetSmartDashboardType ()**

**void InitTable (**ITable** \*subTable)**

Initializes a table for this sendable object.

**ITable \* GetTable ()**

## Detailed Description

Handle operation of the accelerometer.

The accelerometer reads acceleration directly through the sensor. Many sensors have multiple axis and can be treated as multiple devices. Each is calibrated by finding the center value over a period of time.

---

# Constructor & Destructor Documentation

## Accelerometer::Accelerometer ( **UINT32 channel** ) [explicit]

Create a new instance of an accelerometer.

The accelerometer is assumed to be in the first analog module in the given analog channel. The constructor allocates desired analog channel.

## Accelerometer::Accelerometer ( **UINT8 moduleNumber,** **UINT32 channel** )

Create new instance of accelerometer.

Make a new instance of the accelerometer given a module and channel. The constructor allocates the desired analog channel from the specified module

### Parameters:

**moduleNumber** The analog module (1 or 2).  
**channel** The analog channel (1..8)

## Accelerometer::Accelerometer ( **AnalogChannel \* channel** ) [explicit]

Create a new instance of **Accelerometer** from an existing **AnalogChannel**.

Make a new instance of accelerometer given an **AnalogChannel**. This is particularly useful if the port is going to be read as an analog channel as well as through the **Accelerometer** class.

# Member Function Documentation

## **float Accelerometer::GetAcceleration( )**

Return the acceleration in Gs.

The acceleration is returned units of Gs.

### Returns:

The current acceleration of the sensor in Gs.

## **std::string Accelerometer::GetSmartDashboardType( ) [virtual]**

### Returns:

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

## **ITable \* Accelerometer::GetTable( ) [virtual]**

### Returns:

the table that is currently associated with the sendable

Implements **Sendable**.

## **void Accelerometer::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

### Parameters:

**subtable** The table to put the values in.

Implements **Sendable**.

## **double Accelerometer::PIDGet( ) [virtual]**

Get the Acceleration for the PID Source parent.

## Returns:

The current acceleration in Gs.

Implements **PIDSource**.

### **void Accelerometer::SetSensitivity ( float sensitivity )**

Set the accelerometer sensitivity.

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

#### **Parameters:**

**sensitivity** The sensitivity of accelerometer in Volts per G.

### **void Accelerometer::SetZero ( float zero )**

Set the voltage that corresponds to 0 G.

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

#### **Parameters:**

**zero** The zero G voltage.

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

- C:/WindRiver/workspace/WPILib/**Accelerometer.h**
- C:/WindRiver/workspace/WPILib/Accelerometer.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

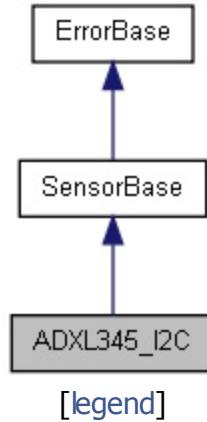
[Classes](#) | [Public Types](#) |  
[Public Member Functions](#) | [Protected Types](#) |  
[Protected Attributes](#) |  
[Static Protected Attributes](#)

# ADXL345\_I2C Class Reference

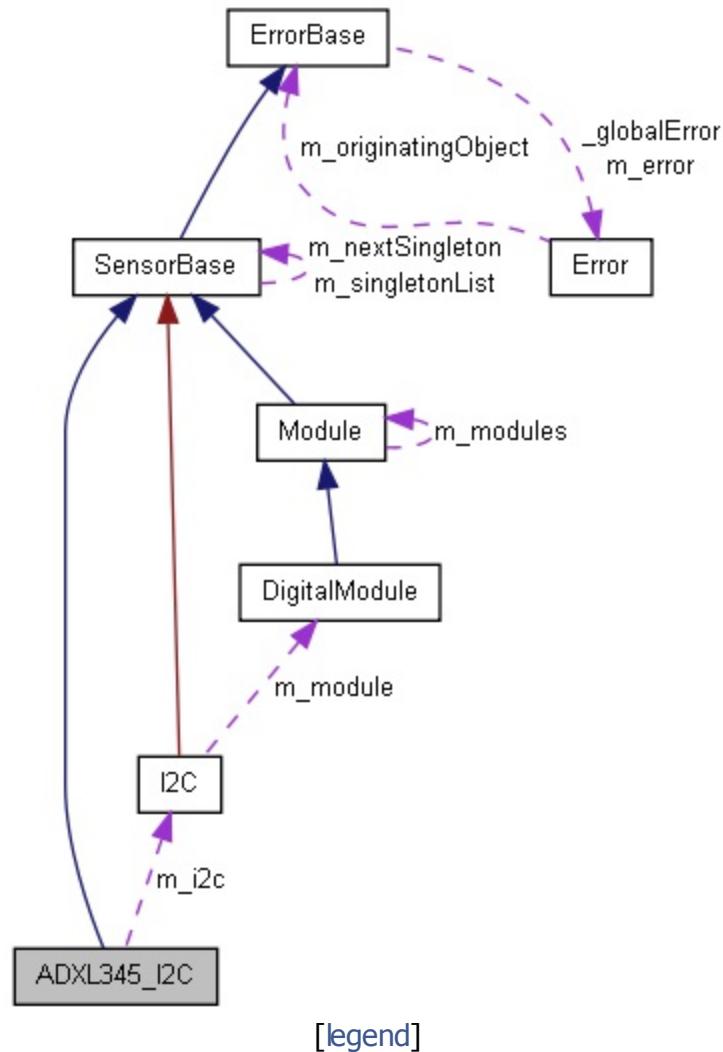
ADXL345 Accelerometer on I2C. More...

```
#include <ADXL345_I2C.h>
```

Inheritance diagram for ADXL345\_I2C:



Collaboration diagram for ADXL345\_I2C:



List of all members.

# Classes

struct **AllAxes**

## Public Types

```
enum DataFormat_Range { kRange_2G = 0x00, kRange_4G =  
    0x01, kRange_8G = 0x02, kRange_16G = 0x03 }  
enum Axes { kAxis_X = 0x00, kAxis_Y = 0x02, kAxis_Z = 0x04 }
```

	<b>ADXL345_I2C</b> (UINT8 moduleNumber, DataFormat_Range range=kRange_2G) Constructor.
virtual	<b>~ADXL345_I2C</b> () Destructor.
virtual double	<b>GetAcceleration</b> (Axes axis) Get the acceleration of one axis in Gs.
virtual AllAxes	<b>GetAccelerations</b> () Get the acceleration of all axes in Gs.

enum	<b>PowerCtlFields</b> { <b>kPowerCtl_Link</b> = 0x20, <b>kPowerCtl_AutoSleep</b> = 0x10, <b>kPowerCtl_Measure</b> = 0x08, <b>kPowerCtl_Sleep</b> = 0x04 }
enum	<b>DataFormatFields</b> { <b>kDataFormat_SelfTest</b> = 0x80, <b>kDataFormat_SPI</b> = 0x40, <b>kDataFormat_IntInvert</b> = 0x20, <b>kDataFormat_FullRes</b> = 0x08, <b>kDataFormat_Justify</b> = 0x04 }

## Protected Attributes

**I2C \* m\_i2c**

static const UINT8 **kAddress** = 0x3A  
static const UINT8 **kPowerCtlRegister** = 0x2D  
static const UINT8 **kDataFormatRegister** = 0x31  
static const UINT8 **kDataRegister** = 0x32

static const double **kGsPerLSB** = 0.00390625

## Detailed Description

### ADXL345 Accelerometer on I2C.

This class allows access to a Analog Devices ADXL345 3-axis accelerometer on an [I2C](#) bus. This class assumes the default (not alternate) sensor address of 0x3A (8-bit address).

---

# Constructor & Destructor Documentation

```
ADXL345_I2C::ADXL345_I2C( UINT8 moduleNumber,  
                            ADXL345_I2C::DataFormat_Range range = kRangeExplicit  
                            ) [explicit]
```

Constructor.

## Parameters:

**moduleNumber** The digital module that the sensor is plugged into (1 or 2).  
**range** The range (+ or -) that the accelerometer will measure.

# Member Function Documentation

## **double ADXL345\_I2C::GetAcceleration ( ADXL345\_I2C::Axes **axis** ) [virtual]**

Get the acceleration of one axis in Gs.

### **Parameters:**

**axis** The axis to read from.

### **Returns:**

Acceleration of the ADXL345 in Gs.

## **ADXL345\_I2C::AllAxes ADXL345\_I2C::GetAccelerations ( ) [virtual]**

Get the acceleration of all axes in Gs.

### **Returns:**

Acceleration measured on all axes of the ADXL345 in Gs.

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

- C:/WindRiver/workspace/WPILib/**ADXL345\_I2C.h**
- C:/WindRiver/workspace/WPILib/ADXL345\_I2C.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

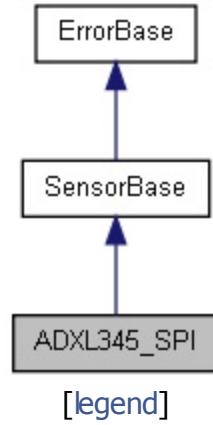
[Classes](#) | [Public Types](#) |  
[Public Member Functions](#) | [Protected Types](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#) |  
[Static Protected Attributes](#)

# ADXL345\_SPI Class Reference

ADXL345 **Accelerometer** on **SPI**. More...

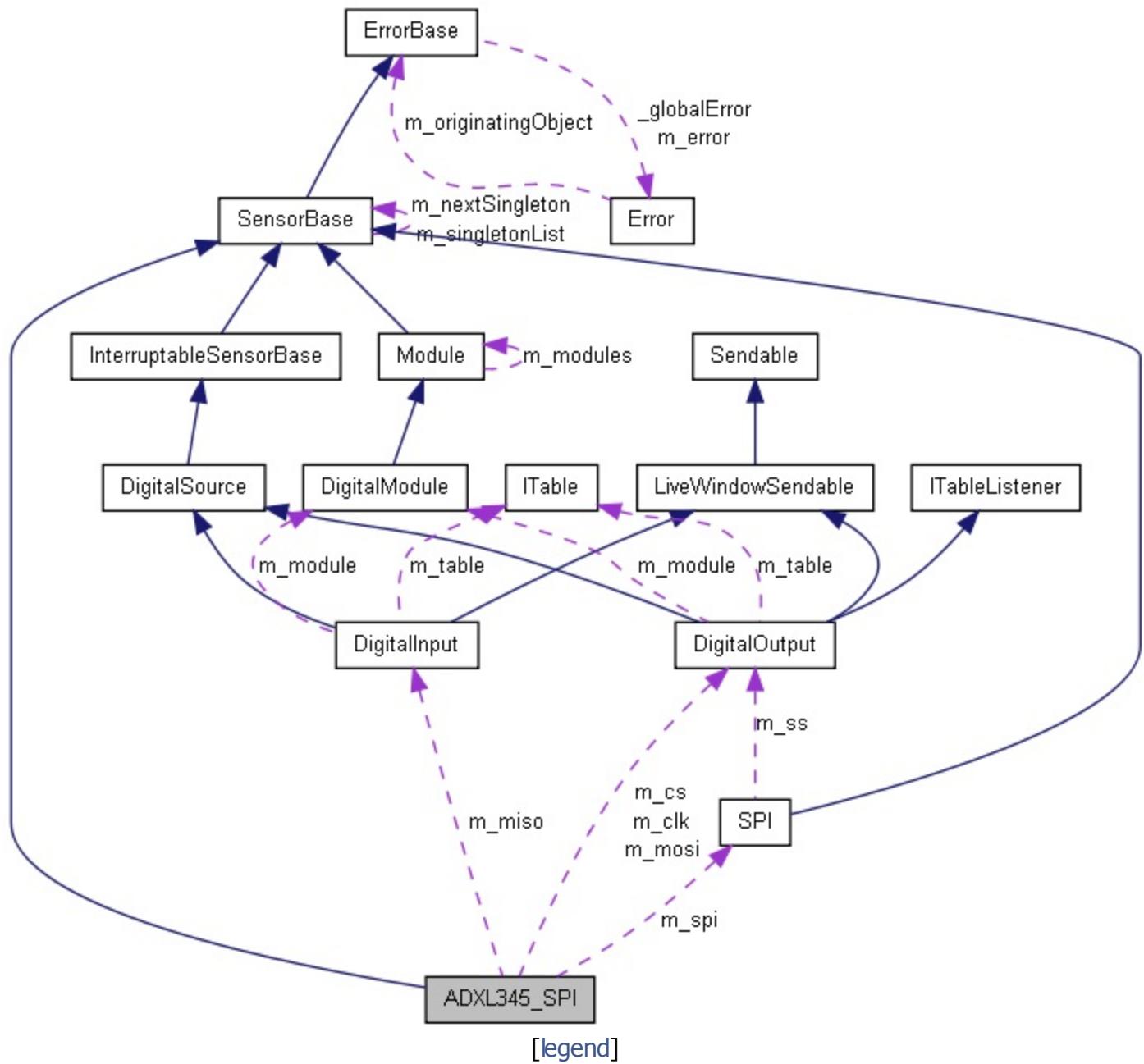
```
#include <ADXL345_SPI.h>
```

Inheritance diagram for ADXL345\_SPI:



[[legend](#)]

Collaboration diagram for ADXL345\_SPI:



List of all members.

# Classes

struct **AllAxes**

enum **DataFormat\_Range** { **kRange\_2G** = 0x00, **kRange\_4G** = 0x01, **kRange\_8G** = 0x02, **kRange\_16G** = 0x03 }  
enum **Axes** { **kAxis\_X** = 0x00, **kAxis\_Y** = 0x02, **kAxis\_Z** = 0x04 }

**ADXL345\_SPI** (**DigitalOutput** &clk, **DigitalOutput** &mosi, **DigitalInput** &miso, **DigitalOutput** &cs, DataFormat\_Range range=kRange\_2G)

Constructor.

**ADXL345\_SPI** (**DigitalOutput** \*clk, **DigitalOutput** \*mosi, **DigitalInput** \*miso, **DigitalOutput** \*cs, DataFormat\_Range range=kRange\_2G)

Constructor.

**ADXL345\_SPI** (UINT8 moduleNumber, UINT32 clk, UINT32 mosi, UINT32 miso, UINT32 cs, DataFormat\_Range range=kRange\_2G)

Constructor.

virtual **~ADXL345\_SPI** ()

Destructor.

virtual double **GetAcceleration** (Axes axis)

Get the acceleration of one axis in Gs.

virtual **AllAxes** **GetAccelerations** ()

enum	<b>SPIAddressFields</b> { <b>kAddress_Read</b> = 0x80, <b>kAddress_MultiByte</b> = 0x40 }
enum	<b>PowerCtlFields</b> { <b>kPowerCtl_Link</b> = 0x20, <b>kPowerCtl_AutoSleep</b> = 0x10, <b>kPowerCtl_Measure</b> = 0x08, <b>kPowerCtl_Sleep</b> = 0x04 }
enum	<b>DataFormatFields</b> { <b>kDataFormat_SelfTest</b> = 0x80, <b>kDataFormat_SPI</b> = 0x40, <b>kDataFormat_IntInvert</b> = 0x20, <b>kDataFormat_FullRes</b> = 0x08, <b>kDataFormat_Justify</b> = 0x04 }

# Protected Member Functions

```
void Init (DigitalOutput *clk, DigitalOutput *mosi, DigitalInput
           *miso, DigitalOutput *cs, DataFormat_Range range)
Internal common init function.
```

**DigitalOutput** \* **m\_clk**

**DigitalOutput** \* **m\_mosi**

**DigitalInput** \* **m\_miso**

**DigitalOutput** \* **m\_cs**

**SPI** \* **m\_spi**

## Static Protected Attributes

---

```
static const UINT8 kPowerCtlRegister = 0x2D
static const UINT8 kDataFormatRegister = 0x31
static const UINT8 kDataRegister = 0x32
static const double kGsPerLSB = 0.00390625
```

---

## Detailed Description

ADXL345 **Accelerometer** on **SPI**.

This class allows access to an Analog Devices ADXL345 3-axis accelerometer via **SPI**. This class assumes the sensor is wired in 4-wire **SPI** mode.

---

# Constructor & Destructor Documentation

```
ADXL345_SPI::ADXL345_SPI ( DigitalOutput & clk,  
                            DigitalOutput & mosi,  
                            DigitalInput & miso,  
                            DigitalOutput & cs,  
                            DataFormat_Range range = kRange_2G  
 )
```

Constructor.

## Parameters:

- clk** The GPIO the clock signal is wired to.
- mosi** The GPIO the MOSI (Master Out Slave In) signal is wired to.
- miso** The GPIO the MISO (Master In Slave Out) signal is wired to.
- cs** The GPIO the CS (Chip Select) signal is wired to.
- range** The range (+ or -) that the accelerometer will measure.

```
ADXL345_SPI::ADXL345_SPI ( DigitalOutput * clk,  
                            DigitalOutput * mosi,  
                            DigitalInput * miso,  
                            DigitalOutput * cs,  
                            DataFormat_Range range = kRange_2G  
 )
```

Constructor.

## Parameters:

- clk** The GPIO the clock signal is wired to.
- mosi** The GPIO the MOSI (Master Out Slave In) signal is wired to.
- miso** The GPIO the MISO (Master In Slave Out) signal is wired to.
- cs** The GPIO the CS (Chip Select) signal is wired to.
- range** The range (+ or -) that the accelerometer will measure.

```
ADXL345_SPI::ADXL345_SPI ( UINT8 moduleNum,  
                            UINT32 clk,  
                            UINT32 mosi,
```

```
    UINT32  
    UINT32  
ADXL345_SPI::DataFormat_Range range = kR  
)
```

Constructor.

#### Parameters:

<b>moduleNumber</b>	The digital module with the sensor attached.
<b>clk</b>	The GPIO the clock signal is wired to.
<b>mosi</b>	The GPIO the MOSI (Master Out Slave In) signal is wired to.
<b>miso</b>	The GPIO the MISO (Master In Slave Out) signal is wired to.
<b>cs</b>	The GPIO the CS (Chip Select) signal is wired to.
<b>range</b>	The range (+ or -) that the accelerometer will measure.

# Member Function Documentation

## **double ADXL345\_SPI::GetAcceleration ( ADXL345\_SPI::Axes **axis** ) [virtual]**

Get the acceleration of one axis in Gs.

### **Parameters:**

**axis** The axis to read from.

### **Returns:**

Acceleration of the ADXL345 in Gs.

## **ADXL345\_SPI::AllAxes ADXL345\_SPI::GetAccelerations ( ) [virtual]**

Get the acceleration of all axes in Gs.

### **Returns:**

Acceleration measured on all axes of the ADXL345 in Gs.

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

- C:/WindRiver/workspace/WPILib/**ADXL345\_SPI.h**
- C:/WindRiver/workspace/WPILib/ADXL345\_SPI.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)[ADXL345\\_SPI](#) › [AllAxes](#) ›

Public Attributes

# **ADXL345\_SPI::AllAxes Struct Reference**

---

List of all members.

## Public Attributes

double **XAxis**

double **YAxis**

double **ZAxis**

---

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

- C:/WindRiver/workspace/WPILib/**ADXL345\_SPI.h**

---

Generated by  1.7.2

[Class List](#)[Class Hierarchy](#)[Class Members](#)[ADXL345\\_I2C](#) › [AllAxes](#) ›

Public Attributes

# **ADXL345\_I2C::AllAxes Struct Reference**

---

List of all members.

## Public Attributes

double **XAxis**

double **YAxis**

double **ZAxis**

---

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

- C:/WindRiver/workspace/WPILib/**ADXL345\_I2C.h**

---

Generated by  1.7.2

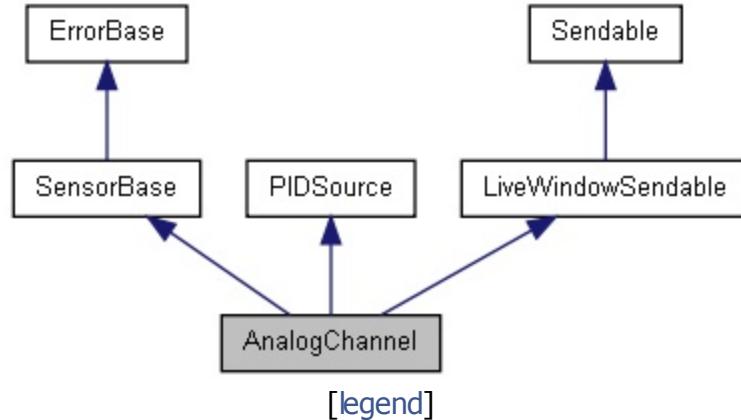


# AnalogChannel Class Reference

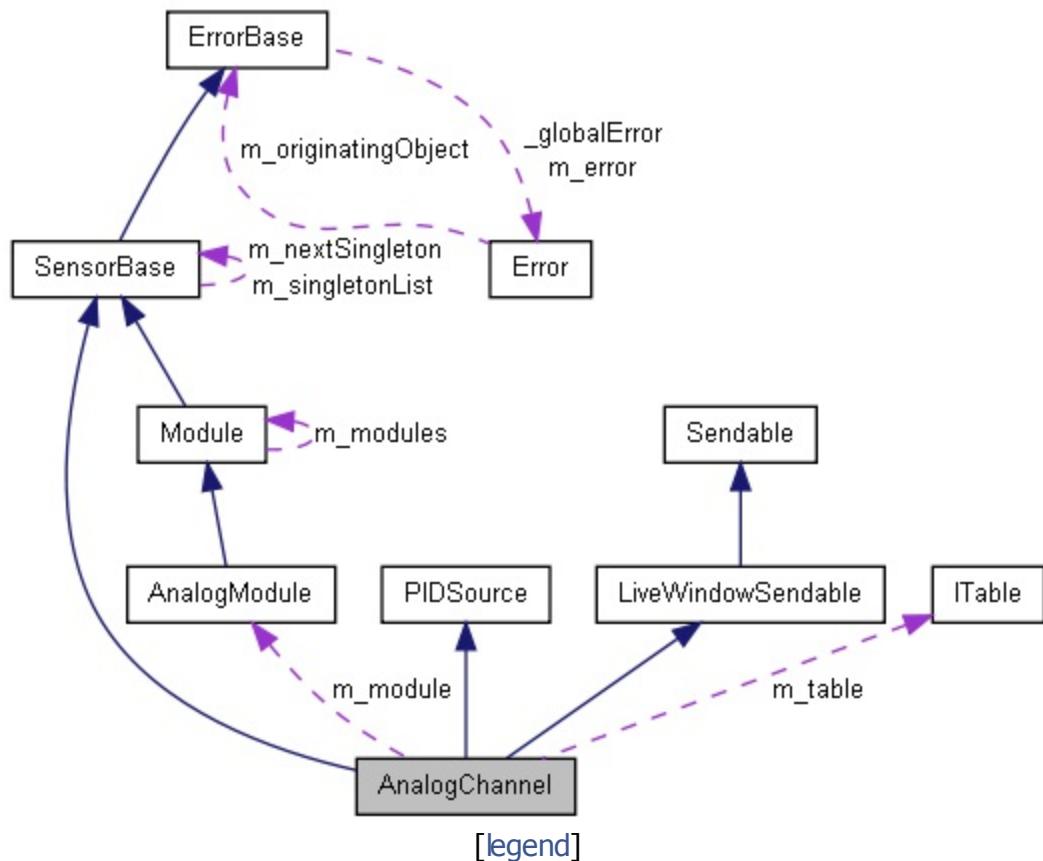
Analog channel class. More...

```
#include <AnalogChannel.h>
```

Inheritance diagram for AnalogChannel:



Collaboration diagram for AnalogChannel:



List of all members.

# Public Member Functions

**AnalogChannel** (UINT8 moduleNumber, UINT32 channel)  
Construct an analog channel on a specified module.

**AnalogChannel** (UINT32 channel)

Construct an analog channel on the default module.

virtual **~AnalogChannel ()**

Channel destructor.

**AnalogModule** \* **GetModule ()**

Get the analog module that this channel is on.

INT16 **GetValue ()**

Get a sample straight from this channel on the module.

INT32 **GetAverageValue ()**

Get a sample from the output of the oversample and average engine for this channel.

float **GetVoltage ()**

Get a scaled sample straight from this channel on the module.

float **GetAverageVoltage ()**

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

UINT8 **GetModuleNumber ()**

Get the module number.

UINT32 **GetChannel ()**

Get the channel number.

void **SetAverageBits** (UINT32 bits)

Set the number of averaging bits.

UINT32 **GetAverageBits ()**

Get the number of averaging bits previously configured.

void **SetOversampleBits** (UINT32 bits)

Set the number of oversample bits.

UINT32 **GetOversampleBits ()**

Get the number of oversample bits previously configured.

UINT32 **GetLSBWeight ()**

Get the factory scaling least significant bit weight constant.

INT32 **GetOffset ()**

Get the factory scaling offset constant.

bool **IsAccumulatorChannel ()**

Is the channel attached to an accumulator.

void	<b>InitAccumulator ()</b>	Initialize the accumulator.
void	<b>SetAccumulatorInitialValue (INT64 value)</b>	Set an initial value for the accumulator.
void	<b>ResetAccumulator ()</b>	Resets the accumulator to the initial value.
void	<b>SetAccumulatorCenter (INT32 center)</b>	Set the center value of the accumulator.
void	<b>SetAccumulatorDeadband (INT32 deadband)</b>	Set the accumulator's deadband.
INT64	<b>GetAccumulatorValue ()</b>	Read the accumulated value.
UINT32	<b>GetAccumulatorCount ()</b>	Read the number of accumulated values.
void	<b>GetAccumulatorOutput (INT64 *value, UINT32 *count)</b>	Read the accumulated value and the number of accumulated values atomically.
double	<b>PIDGet ()</b>	Get the Average voltage for the PID Source base object.
void	<b>UpdateTable ()</b>	Update the table for this sendable object with the latest values.
void	<b>StartLiveWindowMode ()</b>	Start having this sendable object automatically respond to value changes reflect the value on the table.
void	<b>StopLiveWindowMode ()</b>	Stop having this sendable object automatically respond to value changes.
std::string	<b>GetSmartDashboardType ()</b>	
void	<b>InitTable (ITable *subTable)</b>	Initializes a table for this sendable object.
<b>ITable *</b>	<b>GetTable ()</b>	

## Static Public Attributes

```
static const UINT8 kAccumulatorModuleNumber = 1
```

```
static const UINT32 kAccumulatorNumChannels = 2
```

```
static const UINT32 kAccumulatorChannels [kAccumulatorNumChannels] = {1, 2}
```

## Detailed Description

Analog channel class.

Each analog channel is read from hardware as a 12-bit number representing -10V to 10V.

Connected to each analog channel is an averaging and oversampling engine. This engine accumulates the specified ( by [SetAverageBits\(\)](#) and [SetOversampleBits\(\)](#) ) number of samples before returning a new value. This is not a sliding window average. The only difference between the oversampled samples and the averaged samples is that the oversampled samples are simply accumulated effectively increasing the resolution, while the averaged samples are divided by the number of samples to retain the resolution, but get more stable values.

---

# Constructor & Destructor Documentation

```
AnalogChannel::AnalogChannel ( UINT8 moduleNumber,
                                UINT32 channel
                            )
```

Construct an analog channel on a specified module.

## Parameters:

<b>moduleNumber</b>	The analog module (1 or 2).
<b>channel</b>	The channel number to represent.

```
AnalogChannel::AnalogChannel ( UINT32 channel ) [explicit]
```

Construct an analog channel on the default module.

## Parameters:

<b>channel</b>	The channel number to represent.
----------------	----------------------------------

# Member Function Documentation

## **UINT32 AnalogChannel::GetAccumulatorCount( )**

Read the number of accumulated values.

Read the count of the accumulated values since the accumulator was last Reset().

### **Returns:**

The number of times samples from the channel were accumulated.

## **void AnalogChannel::GetAccumulatorOutput( INT64 \* **value**,                                   UINT32 \* **count** )**

Read the accumulated value and the number of accumulated values atomically.

This function reads the value and count from the FPGA atomically. This can be used for averaging.

### **Parameters:**

**value** Pointer to the 64-bit accumulated output.

**count** Pointer to the number of accumulation cycles.

## **INT64 AnalogChannel::GetAccumulatorValue( )**

Read the accumulated value.

Read the value that has been accumulating on channel 1. The accumulator is attached after the oversample and average engine.

### **Returns:**

The 64-bit value accumulated since the last Reset().

## **UINT32 AnalogChannel::GetAverageBits( )**

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  $2^{**\text{bits}}$ . The averaging is done automatically in the FPGA.

**Returns:**

Number of bits of averaging previously configured.

## **INT32 AnalogChannel::GetAverageValue( )**

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  $2^{**}$  (`OversampleBits + AverageBits`) samples have been acquired from the module on this channel. Use [GetAverageVoltage\(\)](#) to get the analog value in calibrated units.

**Returns:**

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

## **float AnalogChannel::GetAverageVoltage( )**

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.

**Returns:**

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

## **UINT32 AnalogChannel::GetChannel( )**

Get the channel number.

**Returns:**

The channel number.

## **UINT32 AnalogChannel::GetLSBWeight( )**

Get the factory scaling least significant bit weight constant.

The least significant bit weight constant for the channel that was calibrated in manufacturing and stored in an eeprom in the module.

$$\text{Volts} = ((\text{LSB\_Weight} * 1\text{e}-9) * \text{raw}) - (\text{Offset} * 1\text{e}-9)$$

**Returns:**

Least significant bit weight.

## **AnalogModule \* AnalogChannel::GetModule( )**

Get the analog module that this channel is on.

**Returns:**

A pointer to the **AnalogModule** that this channel is on.

## **UINT8 AnalogChannel::GetModuleNumber( )**

Get the module number.

**Returns:**

The module number.

## **INT32 AnalogChannel::GetOffset( )**

Get the factory scaling offset constant.

The offset constant for the channel that was calibrated in manufacturing and stored in an eeprom in the module.

$$\text{Volts} = ((\text{LSB\_Weight} * 1\text{e}-9) * \text{raw}) - (\text{Offset} * 1\text{e}-9)$$

**Returns:**

Offset constant.

## **UINT32 AnalogChannel::GetOversampleBits( )**

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  $2^{*\text{bits}}$ . The oversampling is done automatically in the FPGA.

**Returns:**

Number of bits of oversampling previously configured.

**std::string AnalogChannel::GetSmartDashboardType( ) [virtual]****Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

**ITable \* AnalogChannel::GetTable( ) [virtual]****Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

**INT16 AnalogChannel::GetValue( )**

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.

**Returns:**

A sample straight from this channel on the module.

**float AnalogChannel::GetVoltage( )**

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()**.

**Returns:**

A scaled sample straight from this channel on the module.

## **void AnalogChannel::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

## **bool AnalogChannel::IsAccumulatorChannel( )**

Is the channel attached to an accumulator.

### **Returns:**

The analog channel is attached to an accumulator.

## **double AnalogChannel::PIDGet( ) [virtual]**

Get the Average voltage for the PID Source base object.

### **Returns:**

The average voltage.

Implements **PIDSource**.

## **void AnalogChannel::SetAccumulatorCenter( INT32 center )**

Set the center value of the accumulator.

The center value is subtracted from each A/D value before it is added to the accumulator. This is used for the center value of devices like gyros and accelerometers to make integration work and to take the device offset into account when integrating.

This center value is based on the output of the oversampled and averaged source from channel 1. Because of this, any non-zero oversample bits will affect the size of the value for this field.

## **void AnalogChannel::SetAccumulatorInitialValue( INT64 initialValue )**

Set an initial value for the accumulator.

This will be added to all values returned to the user.

### Parameters:

**initialValue** The value that the accumulator should start from when reset.

## **void AnalogChannel::SetAverageBits ( **UINT32 bits** )**

Set the number of averaging bits.

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

### Parameters:

**bits** Number of bits of averaging.

## **void AnalogChannel::SetOversampleBits ( **UINT32 bits** )**

Set the number of oversample bits.

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

### Parameters:

**bits** Number of bits of oversampling.

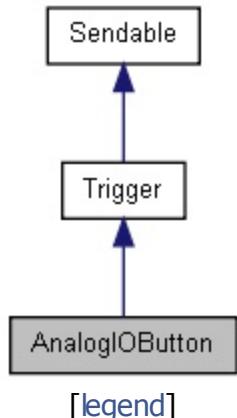
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/**AnalogChannel.h**
- C:/WindRiver/workspace/WPILib/AnalogChannel.cpp

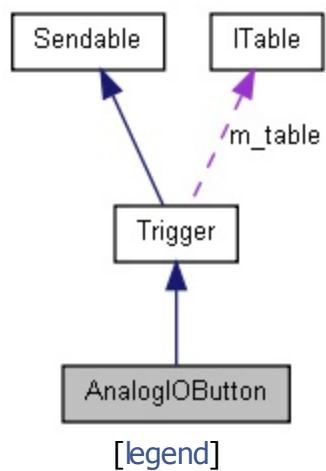


# AnalogIOButton Class Reference

Inheritance diagram for AnalogIOButton:



Collaboration diagram for AnalogIOButton:



List of all members.

# Public Member Functions

**AnalogIOButton** (int port)

virtual bool **Get** ()

---

static const double **kThreshold** = 0.5

---

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

- C:/WindRiver/workspace/WPILib/Buttons/[AnalogIOButton.h](#)
- C:/WindRiver/workspace/WPILib/Buttons/AnalogIOButton.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

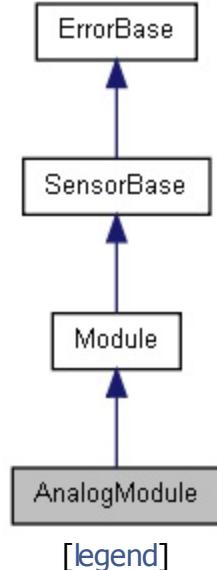
[Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Static Public Attributes](#) |  
[Protected Member Functions](#) | [Friends](#)

# AnalogModule Class Reference

Analog **Module** class. More...

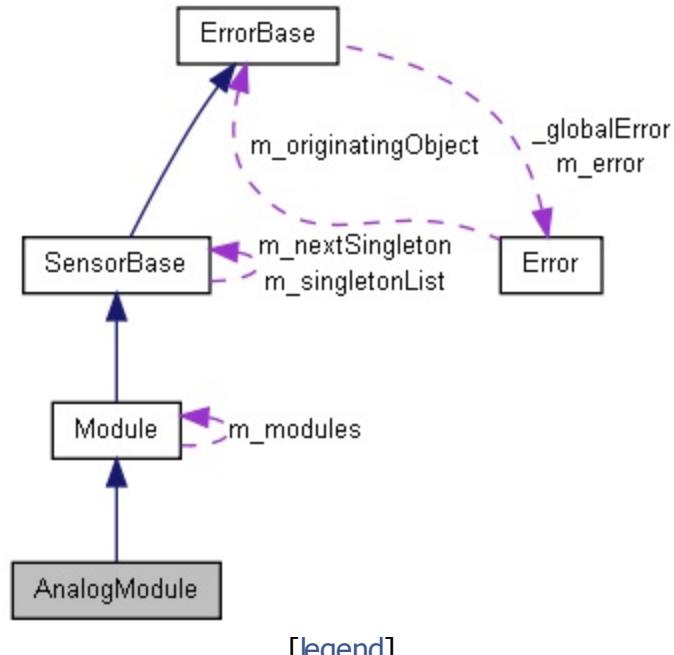
```
#include <AnalogModule.h>
```

Inheritance diagram for AnalogModule:



[legend]

Collaboration diagram for AnalogModule:



[legend]

List of all members.

# Public Member Functions

void	<b>SetSampleRate</b> (float samplesPerSecond) Set the sample rate on the module.
float	<b>GetSampleRate</b> () Get the current sample rate on the module.
void	<b>SetAverageBits</b> (UINT32 channel, UINT32 bits) Set the number of averaging bits.
UINT32	<b>GetAverageBits</b> (UINT32 channel) Get the number of averaging bits.
void	<b>SetOversampleBits</b> (UINT32 channel, UINT32 bits) Set the number of oversample bits.
UINT32	<b>GetOversampleBits</b> (UINT32 channel) Get the number of oversample bits.
INT16	<b>GetValue</b> (UINT32 channel) Get a sample straight from the channel on this module.
INT32	<b>GetAverageValue</b> (UINT32 channel) Get a sample from the output of the oversample and average engine for the channel.
float	<b>GetAverageVoltage</b> (UINT32 channel) Get a scaled sample from the output of the oversample and average engine for the channel.
float	<b>GetVoltage</b> (UINT32 channel) Get a scaled sample straight from the channel on this module.
UINT32	<b>GetLSBWeight</b> (UINT32 channel) Get the factory scaling least significant bit weight constant.
INT32	<b>GetOffset</b> (UINT32 channel) Get the factory scaling offset constant.
INT32	<b>VoltsToValue</b> (INT32 channel, float voltage) Convert a voltage to a raw value for a specified channel.

# Static Public Member Functions

```
static AnalogModule * GetInstance (UINT8 moduleNumber)  
Get an instance of an Analog Module.
```

```
static const long kTimebase = 40000000  
40 MHz clock
```

```
static const long kDefaultOversampleBits = 0
```

```
static const long kDefaultAverageBits = 7
```

```
static const float kDefaultSampleRate = 50000.0
```

```
AnalogModule (UINT8 moduleNumber)  
Create a new instance of an analog module.
```

```
virtual ~AnalogModule ()  
Destructor for AnalogModule.
```

# **Friends**

class **Module**

---

## Detailed Description

Analog **Module** class.

Each module can independently sample its channels at a configurable rate. There is a 64-bit hardware accumulator associated with channel 1 on each module. The accumulator is attached to the output of the oversample and average engine so that the center value can be specified in higher resolution resulting in less error.

---

# Constructor & Destructor Documentation

## AnalogModule::AnalogModule ( **UINT8 moduleNumber** ) [explicit, protected]

Create a new instance of an analog module.

Create an instance of the analog module object. Initialize all the parameters to reasonable values on start. Setting a global value on an analog module can be done only once unless subsequent values are set the previously set value. Analog modules are a singleton, so the constructor is never called outside of this class.

### Parameters:

**moduleNumber** The analog module to create (1 or 2).

# Member Function Documentation

## **UINT32 AnalogModule::GetAverageBits ( UINT32 channel )**

Get the number of averaging bits.

This gets the number of averaging bits from the FPGA. The actual number of averaged samples is  $2^{**\text{bits}}$ . The averaging is done automatically in the FPGA.

### **Parameters:**

**channel** Channel to address.

### **Returns:**

Bits to average.

## **INT32 AnalogModule::GetAverageValue ( UINT32 channel )**

Get a sample from the output of the oversample and average engine for the channel.

The sample is 12-bit + the value configured in [SetOversampleBits\(\)](#). The value configured in [SetAverageBits\(\)](#) will cause this value to be averaged  $2^{**\text{bits}}$  number of samples. This is not a sliding window. The sample will not change until  $2^{**(\text{OversampleBits} + \text{AverageBits})}$  samples have been acquired from the module on this channel. Use [GetAverageVoltage\(\)](#) to get the analog value in calibrated units.

### **Parameters:**

**channel** Channel number to read.

### **Returns:**

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

## **float AnalogModule::GetAverageVoltage ( UINT32 channel )**

Get a scaled sample from the output of the oversample and average engine for the 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 to read.

**Returns:**

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

**AnalogModule \* AnalogModule::GetInstance ( UINT8 moduleNumber ) [static]**

Get an instance of an Analog **Module**.

Singleton analog module creation where a module is allocated on the first use and the same module is returned on subsequent uses.

**Parameters:**

**moduleNumber** The analog module to get (1 or 2).

**Returns:**

A pointer to the **AnalogModule**.

**UINT32 AnalogModule::GetLSBWeight ( UINT32 channel )**

Get the factory scaling least significant bit weight constant.

The least significant bit weight constant for the channel that was calibrated in manufacturing and stored in an eeprom in the module.

$$\text{Volts} = ((\text{LSB\_Weight} * 1e-9) * \text{raw}) - (\text{Offset} * 1e-9)$$

**Parameters:**

**channel** The channel to get calibration data for.

**Returns:**

Least significant bit weight.

**INT32 AnalogModule::GetOffset ( UINT32 channel )**

Get the factory scaling offset constant.

The offset constant for the channel that was calibrated in manufacturing and stored in an eeprom in the module.

`Volts = ((LSB_Weight * 1e-9) * raw) - (Offset * 1e-9)`

**Parameters:**

**channel** The channel to get calibration data for.

**Returns:**

Offset constant.

## **UINT32 AnalogModule::GetOversampleBits ( UINT32 channel )**

Get the number of oversample bits.

This gets the number of oversample bits from the FPGA. The actual number of oversampled values is  $2^{*\text{bits}}$ . The oversampling is done automatically in the FPGA.

**Parameters:**

**channel** Channel to address.

**Returns:**

Bits to oversample.

## **float AnalogModule::GetSampleRate ( )**

Get the current sample rate on the module.

This assumes one entry in the scan list. This is a global setting for the module and effects all channels.

**Returns:**

Sample rate.

## **INT16 AnalogModule::GetValue ( UINT32 channel )**

Get a sample straight from the channel on this 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.

**Returns:**

A sample straight from the channel on this module.

## **float AnalogModule::GetVoltage ( UINT32 channel )**

Get a scaled sample straight from the channel on this module.

The value is scaled to units of Volts using the calibrated scaling data from [GetLSBWeight\(\)](#) and [GetOffset\(\)](#).

### **Parameters:**

**channel** The channel to read.

### **Returns:**

A scaled sample straight from the channel on this module.

## **void AnalogModule::SetAverageBits ( UINT32 channel,                                   UINT32 bits                                   )**

Set the number of averaging bits.

This sets the number of averaging bits. The actual number of averaged samples is  $2^{**\text{bits}}$ . 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** Analog channel to configure.

**bits** Number of bits to average.

## **void AnalogModule::SetOversampleBits ( UINT32 channel,                                   UINT32 bits                                   )**

Set the number of oversample bits.

This sets the number of oversample bits. The actual number of oversampled values is  $2^{**\text{bits}}$ . 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** Analog channel to configure.

**bits** Number of bits to oversample.

## **void AnalogModule::SetSampleRate ( float samplesPerSecond )**

Set the sample rate on the module.

This is a global setting for the module and effects all channels.

### **Parameters:**

**samplesPerSecond** The number of samples per channel per second.

## **INT32 AnalogModule::VoltsToValue ( INT32 channel, float voltage )**

Convert a voltage to a raw value for a specified channel.

This process depends on the calibration of each channel, so the channel must be specified.

### **Todo:**

This assumes raw values. Oversampling not supported as is.

### **Parameters:**

**channel** The channel to convert for.

**voltage** The voltage to convert.

### **Returns:**

The raw value for the channel.

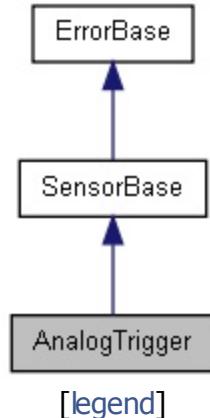
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPIlib/[AnalogModule.h](#)
- C:/WindRiver/workspace/WPIlib/AnalogModule.cpp

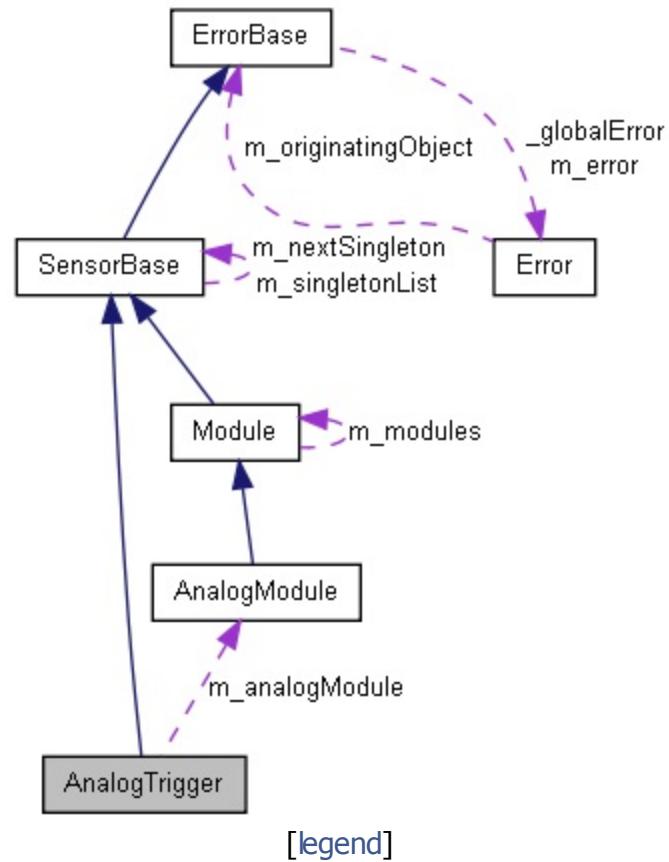


# AnalogTrigger Class Reference

Inheritance diagram for AnalogTrigger:



Collaboration diagram for AnalogTrigger:



List of all members.

# Public Member Functions

**AnalogTrigger** (UINT8 moduleNumber, UINT32 channel)

Constructor for an analog trigger given both the slot and channel.

**AnalogTrigger** (UINT32 channel)

Constructor for an analog trigger given a channel number.

**AnalogTrigger** (**AnalogChannel** \*channel)

Construct an analog trigger given an analog channel.

void **SetLimitsVoltage** (float lower, float upper)

Set the upper and lower limits of the analog trigger.

void **SetLimitsRaw** (INT32 lower, INT32 upper)

Set the upper and lower limits of the analog trigger.

void **SetAveraged** (bool useAveragedValue)

Configure the analog trigger to use the averaged vs.

void **SetFiltered** (bool useFilteredValue)

Configure the analog trigger to use a filtered value.

UINT32 **GetIndex** ()

Return the index of the analog trigger.

bool **GetInWindow** ()

Return the InWindow output of the analog trigger.

bool **GetTriggerState** ()

Return the TriggerState output of the analog trigger.

**AnalogTriggerOutput** \* **CreateOutput** (AnalogTriggerOutput::Type type)

Creates an **AnalogTriggerOutput** object.

# Friends

class **AnalogTriggerOutput**

---

# Constructor & Destructor Documentation

```
AnalogTrigger::AnalogTrigger( UINT8 moduleNumber,  
                           UINT32 channel )
```

Constructor for an analog trigger given both the slot and channel.

## Parameters:

**moduleNumber** The analog module (1 or 2).  
**channel** The analog channel (1..8).

```
AnalogTrigger::AnalogTrigger( UINT32 channel ) [explicit]
```

Constructor for an analog trigger given a channel number.

The default module is used in this case.

## Parameters:

**channel** The analog channel (1..8).

```
AnalogTrigger::AnalogTrigger( AnalogChannel * channel ) [explicit]
```

Construct an analog trigger given an analog channel.

This should be used in the case of sharing an analog channel between the trigger and an analog input object.

# Member Function Documentation

## **AnalogTriggerOutput \* AnalogTrigger::CreateOutput ( AnalogTriggerOutput::Type type )**

Creates an **AnalogTriggerOutput** object.

Gets an output object that can be used for routing. Caller is responsible for deleting the **AnalogTriggerOutput** object.

### **Parameters:**

**type** An enum of the type of output object to create.

### **Returns:**

A pointer to a new **AnalogTriggerOutput** object.

## **UINT32 AnalogTrigger::GetIndex ( )**

Return the index of the analog trigger.

This is the FPGA index of this analog trigger instance.

### **Returns:**

The index of the analog trigger.

## **bool AnalogTrigger::GetInWindow ( )**

Return the InWindow output of the analog trigger.

True if the analog input is between the upper and lower limits.

### **Returns:**

The InWindow output of the analog trigger.

## **bool AnalogTrigger::GetTriggerState ( )**

Return the TriggerState output of the analog trigger.

True if above upper limit. False if below lower limit. If in Hysteresis, maintain previous state.

## Returns:

The TriggerState output of the analog trigger.

### **void AnalogTrigger::SetAveraged ( bool useAveragedValue )**

Configure the analog trigger to use the averaged vs.

raw values. If the value is true, then the averaged value is selected for the analog trigger, otherwise the immediate value is used.

### **void AnalogTrigger::SetFiltered ( bool useFilteredValue )**

Configure the analog trigger to use a filtered value.

The analog trigger will operate with a 3 point average rejection filter. This is designed to help with 360 degree pot applications for the period where the pot crosses through zero.

### **void AnalogTrigger::SetLimitsRaw ( INT32 lower, INT32 upper )**

Set the upper and lower limits of the analog trigger.

The limits are given in ADC codes. If oversampling is used, the units must be scaled appropriately.

### **void AnalogTrigger::SetLimitsVoltage ( float lower, float upper )**

Set the upper and lower limits of the analog trigger.

The limits are given as floating point voltage values.

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

- C:/WindRiver/workspace/WPILib/[AnalogTrigger.h](#)



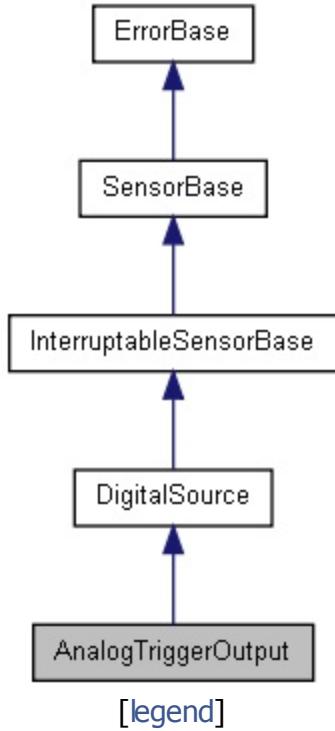
[Public Types](#) | [Public Member Functions](#) |  
[Protected Member Functions](#) | [Friends](#)

# AnalogTriggerOutput Class Reference

Class to represent a specific output from an analog trigger. [More...](#)

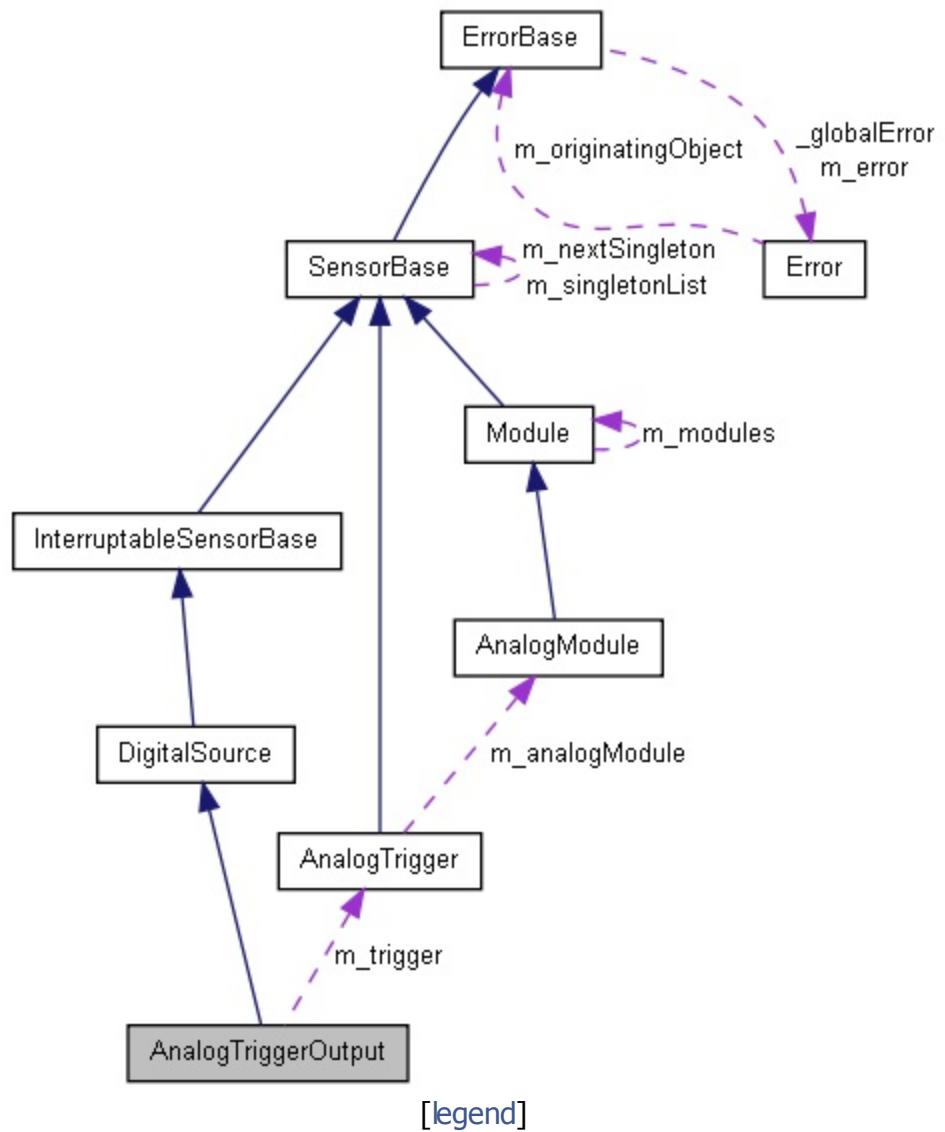
```
#include <AnalogTriggerOutput.h>
```

Inheritance diagram for AnalogTriggerOutput:



[[legend](#)]

Collaboration diagram for AnalogTriggerOutput:



List of all members.

## Public Types

enum **Type** { **kInWindow** = 0, **kState** = 1, **kRisingPulse** = 2, **kFallingPulse** = 3 }

bool **Get** ()

Get the state of the analog trigger output.

virtual UINT32 **GetChannelForRouting** ()

virtual UINT32 **GetModuleForRouting** ()

virtual bool **GetAnalogTriggerForRouting** ()

virtual void **RequestInterrupts** (tInterruptHandler handler, void \*param=NULL)  
Asynchronous handler version.

virtual void **RequestInterrupts** ()

Synchronous Wait version.

# Protected Member Functions

**AnalogTriggerOutput** (**AnalogTrigger** \*trigger, Type outputType)  
Create an object that represents one of the four outputs from an analog trigger.

# Friends

---

class **AnalogTrigger**

---

## Detailed Description

Class to represent a specific output from an analog trigger.

This class is used to get the current output value and also as a [DigitalSource](#) to provide routing of an output to digital subsystems on the FPGA such as [Counter](#), [Encoder](#), and Interrupt.

The TriggerState output indicates the primary output value of the trigger. If the analog signal is less than the lower limit, the output is false. If the analog value is greater than the upper limit, then the output is true. If the analog value is in between, then the trigger output state maintains its most recent value.

The InWindow output indicates whether or not the analog signal is inside the range defined by the limits.

The RisingPulse and FallingPulse outputs detect an instantaneous transition from above the upper limit to below the lower limit, and vice versa. These pulses represent a rollover condition of a sensor and can be routed to an up / down counter or to interrupts. Because the outputs generate a pulse, they cannot be read directly. To help ensure that a rollover condition is not missed, there is an average rejection filter available that operates on the upper 8 bits of a 12 bit number and selects the nearest outlier of 3 samples. This will reject a sample that is (due to averaging or sampling) errantly between the two limits. This filter will fail if more than one sample in a row is errantly in between the two limits. You may see this problem if attempting to use this feature with a mechanical rollover sensor, such as a 360 degree no-stop potentiometer without signal conditioning, because the rollover transition is not sharp / clean enough. Using the averaging engine may help with this, but rotational speeds of the sensor will then be limited.

---

# Constructor & Destructor Documentation

```
AnalogTriggerOutput::AnalogTriggerOutput(AnalogTrigger *  
                                         AnalogTriggerOutput::Type outp  
                                         )  
[pro]
```

Create an object that represents one of the four outputs from an analog trigger.

Because this class derives from [DigitalSource](#), it can be passed into routing functions for [Counter](#), [Encoder](#), etc.

## Parameters:

- trigger** A pointer to the trigger for which this is an output.
- outputType** An enum that specifies the output on the trigger to represent.

# Member Function Documentation

## **bool AnalogTriggerOutput::Get( )**

Get the state of the analog trigger output.

### Returns:

The state of the analog trigger output.

## **bool AnalogTriggerOutput::GetAnalogTriggerForRouting( ) [virtual]**

### Returns:

The value to be written to the module field of a routing mux.

Implements [DigitalSource](#).

## **UINT32 AnalogTriggerOutput::GetChannelForRouting( ) [virtual]**

### Returns:

The value to be written to the channel field of a routing mux.

Implements [DigitalSource](#).

## **UINT32 AnalogTriggerOutput::GetModuleForRouting( ) [virtual]**

### Returns:

The value to be written to the module field of a routing mux.

Implements [DigitalSource](#).

## **void AnalogTriggerOutput::RequestInterrupts( ) [virtual]**

Synchronous Wait version.

Request interrupts synchronously on this analog trigger output.

TODO: Hardware supports interrupts on Analog [Trigger](#) outputs... WPILib should too

Implements [DigitalSource](#).

```
void AnalogTriggerOutput::RequestInterrupts( tInterruptHandler handler,
                                            void * param = NULL
                                          ) [virtual]
```

Asynchronous handler version.

Request interrupts asynchronously on this analog trigger output.

TODO: Hardware supports interrupts on Analog **Trigger** outputs... WPILib should too

Implements **DigitalSource**.

---

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

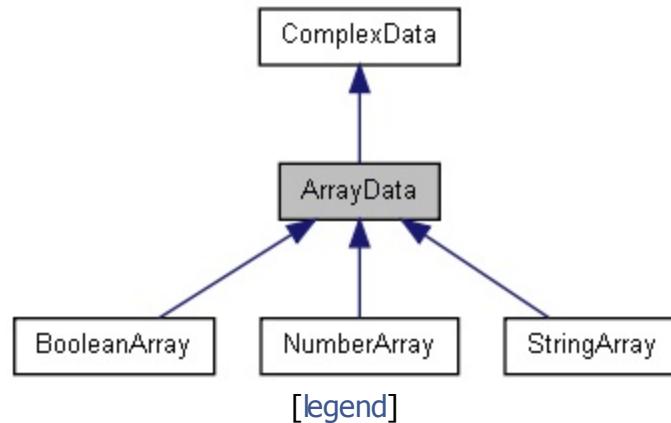
- C:/WindRiver/workspace/WPILib/**AnalogTriggerOutput.h**
- C:/WindRiver/workspace/WPILib/AnalogTriggerOutput.cpp

[Public Member Functions](#) |  
[Protected Member Functions](#) | [Friends](#)

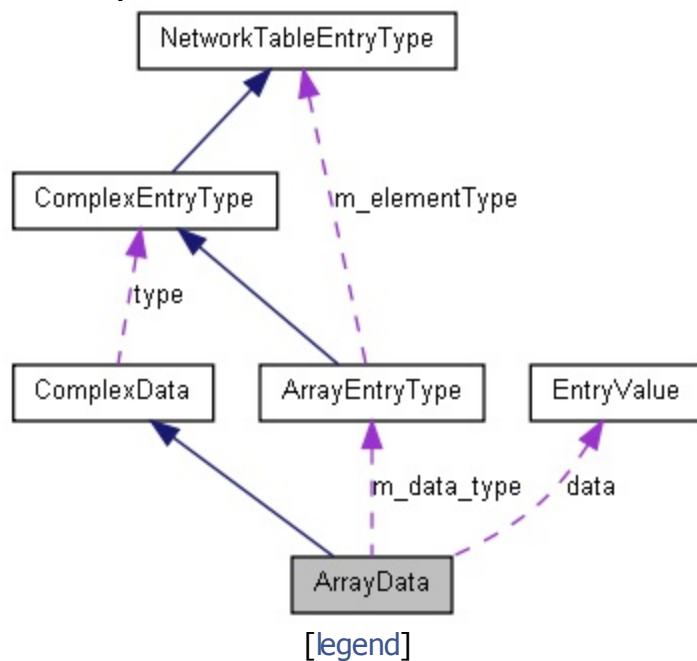
# ArrayData Class Reference

```
#include <ArrayData.h>
```

Inheritance diagram for ArrayData:



Collaboration diagram for ArrayData:



List of all members.

# Public Member Functions

**ArrayData** (**ArrayEntryType** &type)

void **remove** (unsigned int index)

void **setSize** (unsigned int size)

unsigned int **size** ()

**EntryValue** **\_get** (unsigned int index)

void **\_set** (unsigned int index, **EntryValue** value)

void **\_add** (**EntryValue** value)

# Friends

class **ArrayType**

---

# Detailed Description

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/networktables2/type/[ArrayData.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/type/ArrayData.cpp

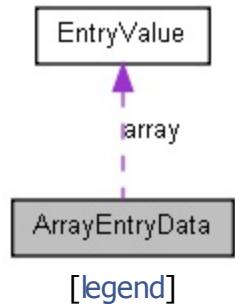
---

Generated by [doxygen](#) 1.7.2



# ArrayEntryData Struct Reference

Collaboration diagram for ArrayEntryData:



List of all members.

## Public Attributes

uint8\_t **length**  
**EntryValue** \* **array**

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

- C:/WindRiver/workspace/WPILib/networktables2/type/[ArrayEntryType.h](#)

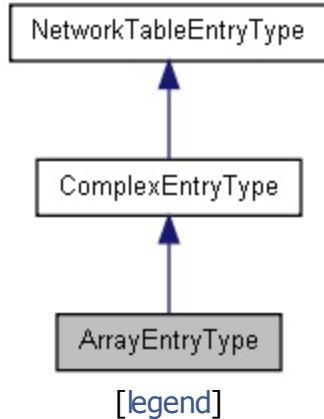
Generated by  1.7.2



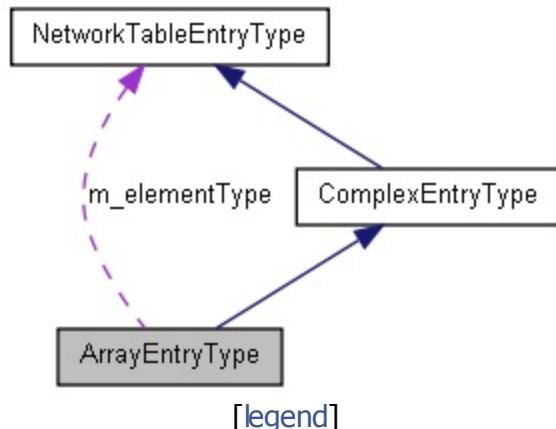
# ArrayType Class Reference

```
#include <ArrayType.h>
```

Inheritance diagram for ArrayType:



Collaboration diagram for ArrayEntryType:



List of all members.

# Public Member Functions

**ArrayEntryType** (TypeId id, **NetworkTableEntryType**  
&elementType)

**EntryValue** **copyElement** (**EntryValue** value)

void **deleteElement** (**EntryValue** value)

void **sendValue** (**EntryValue** value, **DataIOStream** &os)

**EntryValue** **readValue** (**DataIOStream** &is)

**EntryValue** **copyValue** (**EntryValue** value)

void **deleteValue** (**EntryValue** value)

**EntryValue** **internalizeValue** (std::string &key, **ComplexData**  
&externalRepresentation, **EntryValue** currentInternalValue)

void **exportValue** (std::string &key, **EntryValue** internalData,  
**ComplexData** &externalRepresentation)

# Detailed Description

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/networktables2/type/[ArrayType.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/type/ArrayType.cpp

---

Generated by [doxygen](#) 1.7.2

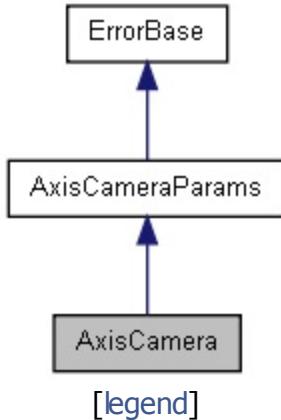
[Public Member Functions](#) |  
[Static Public Member Functions](#)

# AxisCamera Class Reference

[AxisCamera class.](#) More...

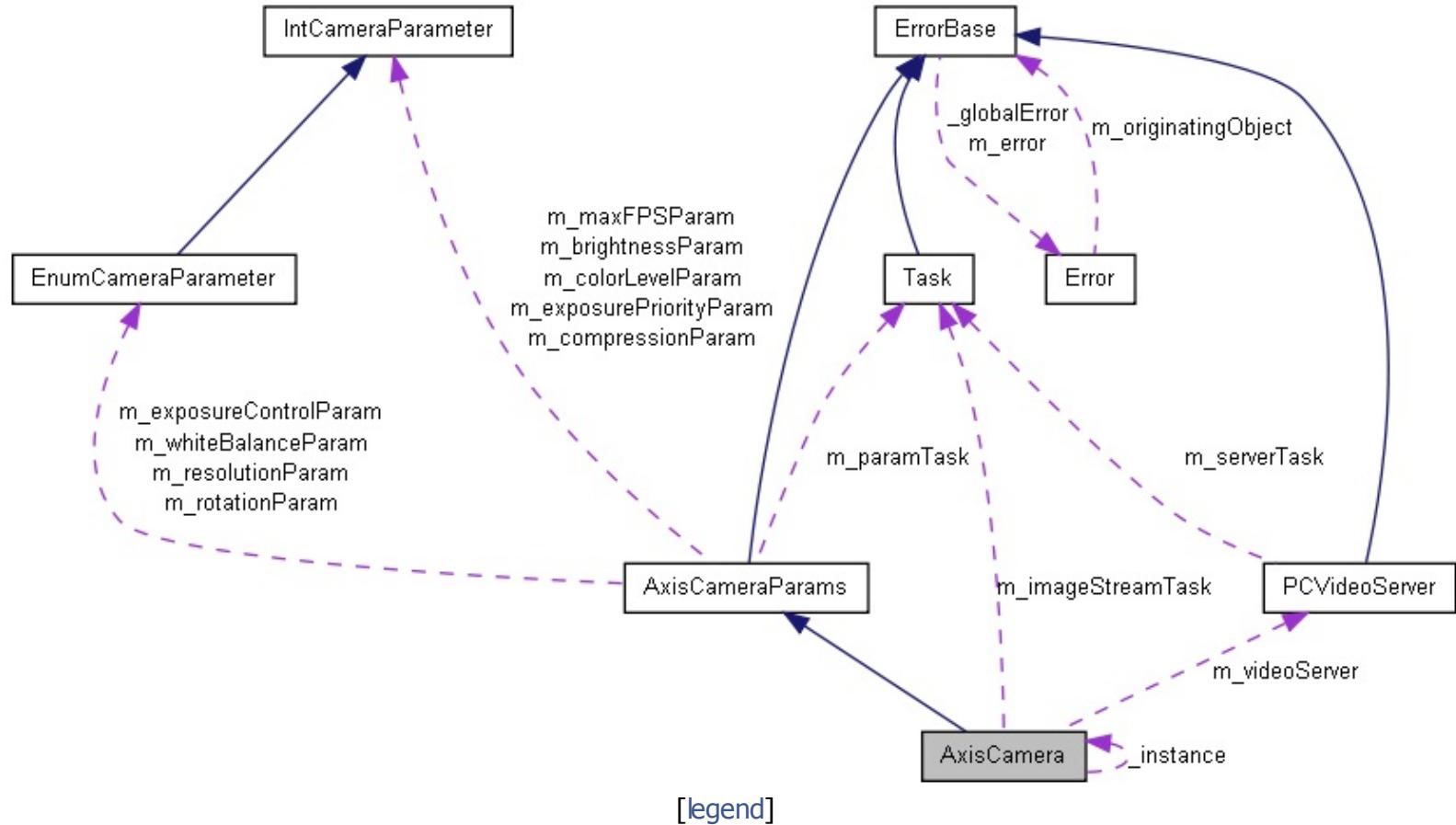
```
#include <AxisCamera.h>
```

Inheritance diagram for AxisCamera:



[legend]

Collaboration diagram for AxisCamera:



[legend]

List of all members.

# Public Member Functions

virtual ~**AxisCamera** ()

Destructor.

bool **IsFreshImage** ()

Return true if the latest image from the camera has not been retrieved by calling **GetImage()** yet.

SEM\_ID **GetNewImageSem** ()

Get the semaphore to be used to synchronize image access with camera acquisition.

int **GetImage** (Image \*imaqImage)

Get an image from the camera and store it in the provided image.

int **GetImage** (ColorImage \*image)

Get an image from the camera and store it in the provided image.

**HSLImage** \* **GetImage** ()

Instantiate a new image object and fill it with the latest image from the camera.

int **CopyJPEG** (char \*\*destImage, int &destImageSize, int

&destImageBufferSize)

Copy an image into an existing buffer.

# Static Public Member Functions

static **AxisCamera** & **GetInstance** (const char \*cameraIP=NULL)

Get a pointer to the **AxisCamera** object, if the object does not exist, create it To use the camera on port 2 of a cRIO-FRC, pass "192.168.0.90" to the first GetInstance call.

static void **DeleteInstance** ()

Called by Java to delete the camera...

## Detailed Description

### **AxisCamera** class.

This class handles everything about the Axis 206 FRC Camera. It starts up 2 tasks each using a different connection to the camera:

- image reading task that reads images repeatedly from the camera
  - parameter handler task in the base class that monitors for changes to parameters and updates the camera
-

# Member Function Documentation

```
int AxisCamera::CopyJPEG ( char ** destImage,
                           int & destImageSize,
                           int & destImageBufferSize
                         )
```

Copy an image into an existing buffer.

This copies an image into an existing buffer rather than creating a new image in memory. That way a new image is only allocated when the image being copied is larger than the destination. This method is called by the **PCVideoServer** class.

## Parameters:

**imageData** The destination image.  
**numBytes** The size of the destination image.

## Returns:

0 if failed (no source image or no memory), 1 if success.

## void AxisCamera::DeleteInstance ( ) [static]

Called by Java to delete the camera...

how thoughtful

## HSLImage \* AxisCamera::GetImage ( )

Instantiate a new image object and fill it with the latest image from the camera.

The returned pointer is owned by the caller and is their responsibility to delete.

## Returns:

a pointer to an **HSLImage** object

## int AxisCamera::GetImage ( Image \* imaqImage )

Get an image from the camera and store it in the provided image.

**Parameters:**

**image** The imaq image to store the result in. This must be an HSL or RGB image This function is called by Java.

**Returns:**

1 upon success, zero on a failure

**int AxisCamera::GetImage ( ColorImage \* image )**

Get an image from the camera and store it in the provided image.

**Parameters:**

**image** The image to store the result in. This must be an HSL or RGB image

**Returns:**

1 upon success, zero on a failure

**AxisCamera & AxisCamera::GetInstance ( const char \* cameraIP = **NULL** ) [static]**

Get a pointer to the **AxisCamera** object, if the object does not exist, create it To use the camera on port 2 of a cRIO-FRC, pass "192.168.0.90" to the first GetInstance call.

**Returns:**

reference to **AxisCamera** object

**SEM\_ID AxisCamera::GetNewImageSem ( )**

Get the semaphore to be used to synchronize image access with camera acquisition.

Call semTake on the returned semaphore to block until a new image is acquired.

The semaphore is owned by the **AxisCamera** class and will be deleted when the class is destroyed.

**Returns:**

A semaphore to notify when new image is received

**bool AxisCamera::IsFreshImage ( )**

Return true if the latest image from the camera has not been retrieved by calling [\*\*GetImage\(\)\*\*](#) yet.

**Returns:**

true if the image has not been retrieved yet.

---

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

- C:/WindRiver/workspace/WPILib/Vision/[\*\*AxisCamera.h\*\*](#)
- C:/WindRiver/workspace/WPILib/Vision/AxisCamera.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

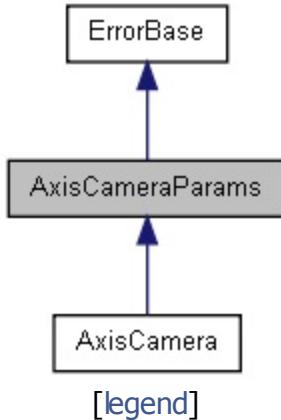
[Public Types](#) | [Public Member Functions](#) |  
[Protected Types](#) |  
[Protected Member Functions](#) |  
[Static Protected Member Functions](#) |  
[Protected Attributes](#)

# AxisCameraParams Class Reference

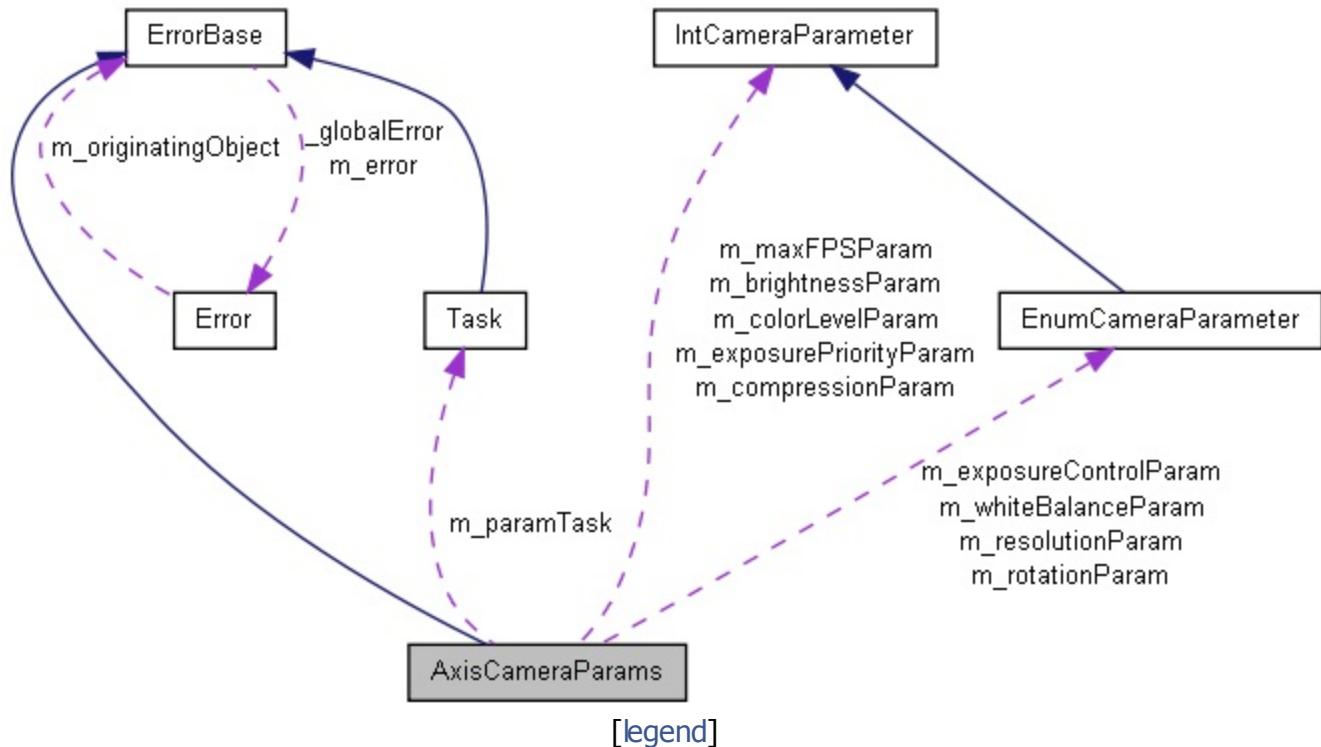
**AxisCameraParams** class. More...

```
#include <AxisCameraParams.h>
```

Inheritance diagram for AxisCameraParams:



Collaboration diagram for AxisCameraParams:



List of all members.

## Public Types

enum	<b>Exposure_t</b> { <b>kExposure_Automatic</b> , <b>kExposure_Hold</b> , <b>kExposure_FlickerFree50Hz</b> , <b>kExposure_FlickerFree60Hz</b> }
enum	<b>WhiteBalance_t</b> { <b>kWhiteBalance_Automatic</b> , <b>kWhiteBalance_Hold</b> , <b>kWhiteBalance_FixedOutdoor1</b> , <b>kWhiteBalance_FixedOutdoor2</b> , <b>kWhiteBalance_FixedIndoor</b> , <b>kWhiteBalance_FixedFlourescent1</b> , <b>kWhiteBalance_FixedFlourescent2</b> }
enum	<b>Resolution_t</b> { <b>kResolution_640x480</b> , <b>kResolution_640x360</b> , <b>kResolution_320x240</b> , <b>kResolution_160x120</b> }
enum	<b>Rotation_t</b> { <b>kRotation_0</b> , <b>kRotation_180</b> }

# Public Member Functions

void **WriteBrightness** (int)

Write the brightness value to the camera.

int **GetBrightness** ()

Get the brightness value.

void **WriteWhiteBalance** (WhiteBalance\_t whiteBalance)

Set the white balance value.

WhiteBalance\_t **GetWhiteBalance** ()

Retrieve the current white balance parameter.

void **WriteColorLevel** (int)

Write the color level to the camera.

int **GetColorLevel** ()

Retrieve the color level from the camera.

void **WriteExposureControl** (Exposure\_t)

Write the exposure control value to the camera.

Exposure\_t **GetExposureControl** ()

Get the exposure value from the camera.

void **WriteExposurePriority** (int)

Write the exposre priority value to the camera.

int **GetExposurePriority** ()

void **WriteMaxFPS** (int)

Write the maximum frames per second that the camera should send Write 0 to send as many as possible.

int **GetMaxFPS** ()

Get the max number of frames per second that the camera will send.

void **WriteResolution** (Resolution\_t)

Write resolution value to camera.

Resolution\_t **GetResolution** ()

Get the resolution value from the camera.  
void **WriteCompression** (int)  
Write the compression value to the camera.

---

int **GetCompression** ()  
Get the compression value from the camera.

---

void **WriteRotation** (Rotation\_t)  
Write the rotation value to the camera.

---

Rotation\_t **GetRotation** ()  
Get the rotation value from the camera.

**AxisCameraParams** (const char \*ipAddress)

**AxisCamera** constructor.

virtual **~AxisCameraParams** ()

Destructor.

virtual void **RestartCameraTask** ()=0

int **CreateCameraSocket** (const char \*requestString)

int **ParamTaskFunction** ()

Main loop of the parameter task.

int **UpdateCamParam** (const char \*param)

Update a camera parameter.

int **ReadCamParams** ()

Read the full param list from camera, use regular expressions to find the bits we care about assign

values to member variables.

static int **s\_ParamTaskFunction** (**AxisCameraParams**  
\*thisPtr)  
Static function to start the parameter updating task.

<b>Task</b>	<b>m_paramTask</b>
<b>UINT32</b>	<b>m_ipAddress</b>
<b>SEM_ID</b>	<b>m_paramChangedSem</b>
<b>SEM_ID</b>	<b>m_socketPossessionSem</b>
<b>IntCameraParameter</b> *	<b>m_brightnessParam</b>
<b>IntCameraParameter</b> *	<b>m_compressionParam</b>
<b>IntCameraParameter</b> *	<b>m_exposurePriorityParam</b>
<b>IntCameraParameter</b> *	<b>m_colorLevelParam</b>
<b>IntCameraParameter</b> *	<b>m_maxFPSParam</b>
<b>EnumCameraParameter</b> *	<b>m_rotationParam</b>
<b>EnumCameraParameter</b> *	<b>m_resolutionParam</b>
<b>EnumCameraParameter</b> *	<b>m_exposureControlParam</b>
<b>EnumCameraParameter</b> *	<b>m_whiteBalanceParam</b>
<b>ParameterVector_t</b>	<b>m_parameters</b>

## Detailed Description

### **AxisCameraParams** class.

This class handles parameter configuration for the Axis 206 Ethernet Camera. It starts up a task with an independent connection to the camera that monitors for changes to parameters and updates the camera. It is only separate from **AxisCamera** to isolate the parameter code from the image streaming code.

---

# Member Function Documentation

## **int AxisCameraParams::GetBrightness( )**

Get the brightness value.

### **Returns:**

Brightness value from the camera.

## **int AxisCameraParams::GetColorLevel( )**

Retrieve the color level from the camera.

the camera color level.

## **int AxisCameraParams::GetCompression( )**

Get the compression value from the camera.

### **Returns:**

The cached compression value from the camera.

## **AxisCameraParams::Exposure\_t AxisCameraParams::GetExposureControl( )**

Get the exposure value from the camera.

### **Returns:**

the exposure value from the camera.

## **int AxisCameraParams::GetMaxFPS( )**

Get the max number of frames per second that the camera will send.

### **Returns:**

Maximum frames per second.

## **AxisCameraParams::Resolution\_t AxisCameraParams::GetResolution( )**

Get the resolution value from the camera.

**Returns:**

resolution value for the camera.

## **AxisCameraParams::Rotation\_t AxisCameraParams::GetRotation( )**

Get the rotation value from the camera.

**Returns:**

The rotation value from the camera (Rotation\_t).

## **AxisCameraParams::WhiteBalance\_t AxisCameraParams::GetWhiteBalance( )**

Retrieve the current white balance parameter.

**Returns:**

The white balance value.

## **int AxisCameraParams::ParamTaskFunction( ) [protected]**

Main loop of the parameter task.

This loop runs continuously checking parameters from the camera for posted changes and updating them if necessary.

## **int AxisCameraParams::UpdateCamParam( const char \* param ) [protected]**

Update a camera parameter.

Write a camera parameter to the camera when it has been changed.

**Parameters:**

**param** the string to insert into the http request.

**Returns:**

0 if it failed, otherwise nonzero.

## **void AxisCameraParams::WriteBrightness ( int brightness )**

Write the brightness value to the camera.

### **Parameters:**

**brightness** valid values 0 .. 100

## **void AxisCameraParams::WriteColorLevel ( int colorLevel )**

Write the color level to the camera.

### **Parameters:**

**colorLevel** valid values are 0 .. 100

## **void AxisCameraParams::WriteCompression ( int compression )**

Write the compression value to the camera.

### **Parameters:**

**compression** Values between 0 and 100.

## **void AxisCameraParams::WriteExposureControl ( Exposure\_t exposureControl )**

Write the exposure control value to the camera.

### **Parameters:**

**exposureControl** A mode to write in the Exposure\_t enum.

## **void AxisCameraParams::WriteExposurePriority ( int exposurePriority )**

Write the exposre priority value to the camera.

### **Parameters:**

**exposurePriority** Valid values are 0, 50, 100. 0 = Prioritize image quality 50 = None 100 = Prioritize frame rate

## **void AxisCameraParams::WriteMaxFPS ( int maxFPS )**

Write the maximum frames per second that the camera should send Write 0 to send as many as possible.

#### Parameters:

**maxFPS** The number of frames the camera should send in a second, exposure permitting.

### **void AxisCameraParams::WriteResolution ( Resolution\_t resolution )**

Write resolution value to camera.

#### Parameters:

**resolution** The camera resolution value to write to the camera. Use the Resolution\_t enum.

### **void AxisCameraParams::WriteRotation ( Rotation\_t rotation )**

Write the rotation value to the camera.

If you mount your camera upside down, use this to adjust the image for you.

#### Parameters:

**rotation** The image from the Rotation\_t enum in **AxisCameraParams** (kRotation\_0 or kRotation\_180)

### **void AxisCameraParams::WriteWhiteBalance ( WhiteBalance\_t whiteBalance )**

Set the white balance value.

#### Parameters:

**whiteBalance** Valid values from the WhiteBalance\_t enum.

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

- C:/WindRiver/workspace/WPILib/Vision/**AxisCameraParams.h**
- C:/WindRiver/workspace/WPILib/Vision/AxisCameraParams.cpp



# BadMessageException Class Reference

---

List of all members.

# Public Member Functions

**BadMessageException** (const char \*message)

const char \* **what** ()

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

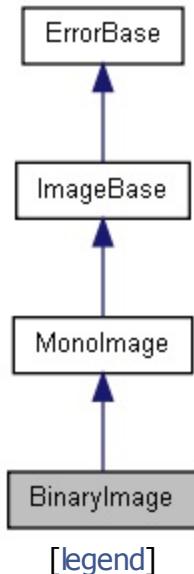
- C:/WindRiver/workspace/WPILib/networktables2/connection/**BadMessageException**.h
- C:/WindRiver/workspace/WPILib/networktables2/connection/BadMessageException.cpp

Generated by  1.7.2

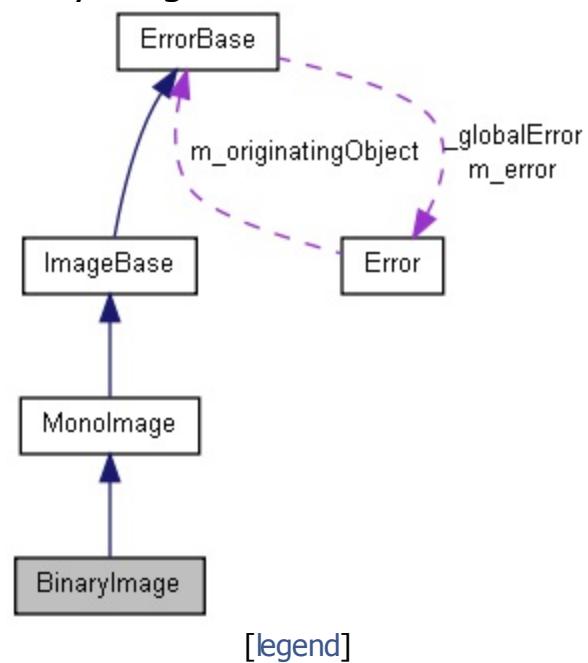


# BinaryImage Class Reference

Inheritance diagram for BinaryImage:



Collaboration diagram for BinaryImage:



List of all members.

# Public Member Functions

	int	<b>GetNumberParticles ()</b>	Get then number of particles for the image.
<b>ParticleAnalysisReport</b>		<b>GetParticleAnalysisReport (int particleNumber)</b>	Get a single particle analysis report.
	void	<b>GetParticleAnalysisReport (int particleNumber, ParticleAnalysisReport *par)</b>	Get a single particle analysis report.
<b>vector&lt; ParticleAnalysisReport &gt; *</b>		<b>GetOrderedParticleAnalysisReports ()</b>	Get an ordered vector of particles for the image.
	<b>BinaryImage *</b>	<b>RemoveSmallObjects (bool connectivity8, int erosions)</b>	
	<b>BinaryImage *</b>	<b>RemoveLargeObjects (bool connectivity8, int erosions)</b>	
	<b>BinaryImage *</b>	<b>ConvexHull (bool connectivity8)</b>	
	<b>BinaryImage *</b>	<b>ParticleFilter (ParticleFilterCriteria2 *criteria, int criteriaCount)</b>	
	virtual void	<b>Write (const char *fileName)</b>	Write a binary image to flash.

# Member Function Documentation

**int BinaryImage::GetNumberParticles( )**

Get then number of particles for the image.

## Returns:

the number of particles found for the image.

```
vector< ParticleAnalysisReport > * BinaryImage::GetOrderedParticleAnalysis
```

Get an ordered vector of particles for the image.

Create a vector of particle analysis reports sorted by size for an image. The vector contains the actual report structures.

## Returns:

a pointer to the vector of particle analysis reports. The caller must delete the vector when finished using it.

**ParticleAnalysisReport** BinaryImage::GetParticleAnalysisReport ( int particleID )

# Get a single particle analysis report.

Get one (of possibly many) particle analysis reports for an image.

## Parameters:

**particleNumber** Which particle analysis report to return.

## Returns:

## the selected particle analysis report

```
void BinaryImage::GetParticleAnalysisReport( int
```

partie

## ParticleAnalysisReport \* par

)

## Get a single particle analysis report.

Get one (of possibly many) particle analysis reports for an image. This version could be more efficient when copying many reports.

## Parameters:

- particleNumber** Which particle analysis report to return.
- par** the selected particle analysis report

## **void BinaryImage::Write ( const char \* **fileName** ) [virtual]**

Write a binary image to flash.

Writes the binary image to flash on the cRIO for later inspection.

## Parameters:

- fileName** the name of the image file written to the flash.

Reimplemented from [ImageBase](#).

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

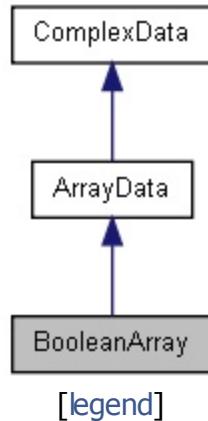
- C:/WindRiver/workspace/WPILib/Vision/[BinaryImage.h](#)
- C:/WindRiver/workspace/WPILib/Vision/BinaryImage.cpp



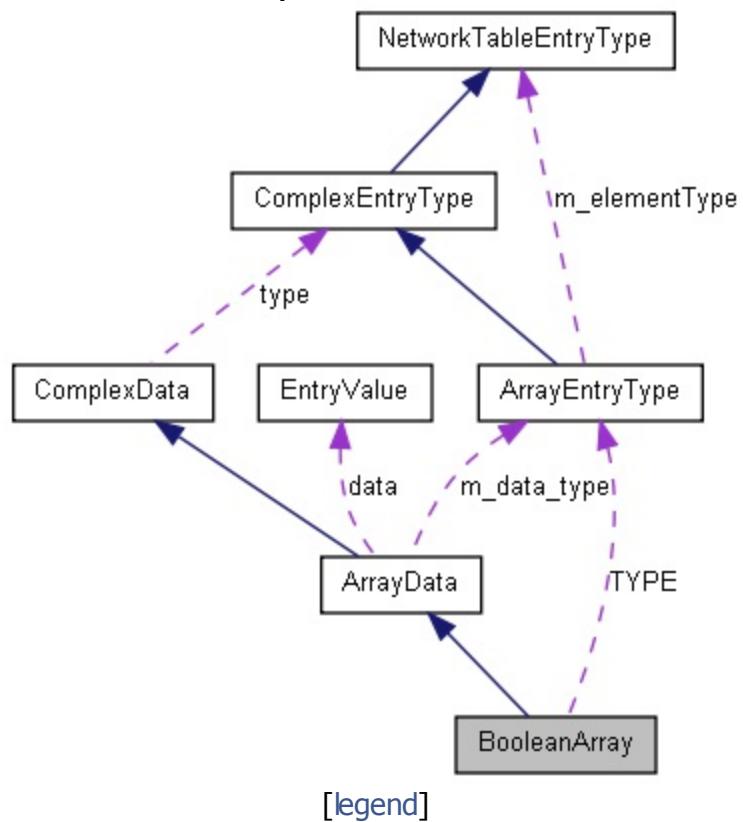
# BooleanArray Class Reference

```
#include <BooleanArray.h>
```

Inheritance diagram for BooleanArray:



Collaboration diagram for BooleanArray:



List of all members.

## Public Member Functions

```
bool get (int index)
void set (int index, bool value)
void add (bool value)
```

```
static const TypeId BOOLEAN_ARRAY_RAW_ID = 0x10
static ArrayEntryType TYPE
```

---

# Detailed Description

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/networktables2/type/**BooleanArray.h**
- C:/WindRiver/workspace/WPILib/networktables2/type/BooleanArray.cpp

---

Generated by [doxygen](#) 1.7.2

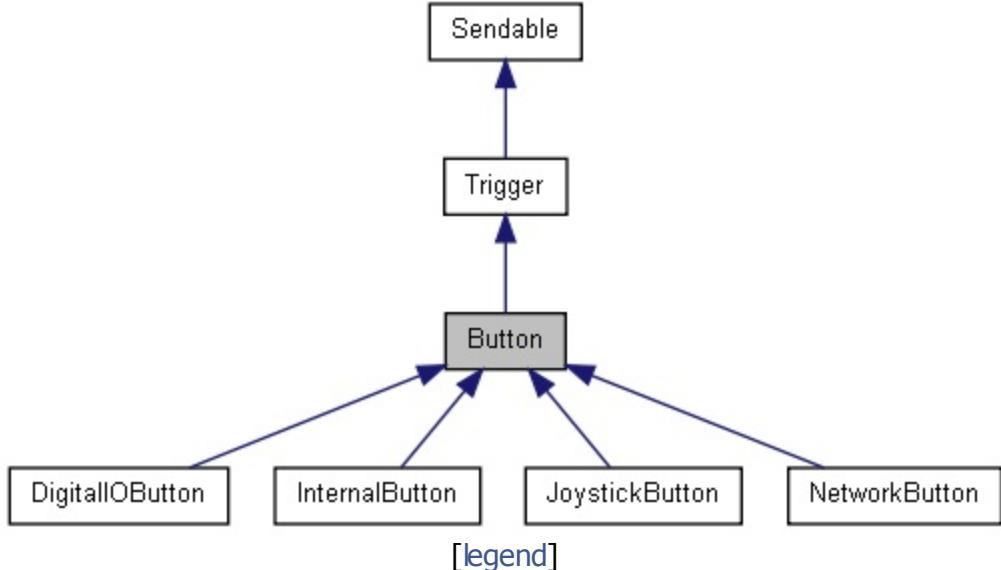


# Button Class Reference

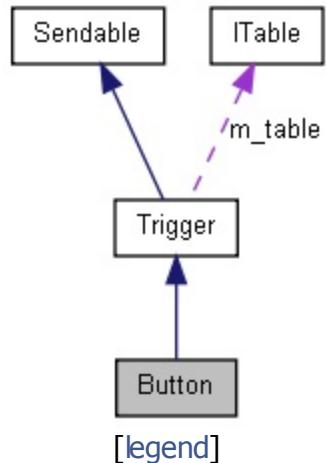
This class provides an easy way to link commands to OI inputs. [More...](#)

```
#include <Button.h>
```

Inheritance diagram for Button:



Collaboration diagram for Button:



List of all members.

## Public Member Functions

virtual void **WhenPressed (Command \*command)**

Specifies the command to run when a button is first pressed.

virtual void **WhileHeld (Command \*command)**

Specifies the command to be scheduled while the button is pressed. The command will be scheduled repeatedly while the button is pressed and will be canceled when the button is released.

virtual void **WhenReleased (Command \*command)**

Specifies the command to run when the button is released. The command will be scheduled a single time.

## Detailed Description

This class provides an easy way to link commands to OI inputs.

It is very easy to link a button to a command. For instance, you could link the trigger button of a joystick to a "score" command.

This class represents a subclass of [Trigger](#) that is specifically aimed at buttons on an operator interface as a common use case of the more generalized [Trigger](#) objects. This is a simple wrapper around [Trigger](#) with the method names renamed to fit the [Button](#) object use.

### Author:

brad

---

# Member Function Documentation

## **void Button::WhenPressed ( Command \* command ) [virtual]**

Specifies the command to run when a button is first pressed.

### Parameters:

**command** The pointer to the command to run

## **void Button::WhenReleased ( Command \* command ) [virtual]**

Specifies the command to run when the button is released The command will be scheduled a single time.

### Parameters:

**The** pointer to the command to run

## **void Button::WhileHeld ( Command \* command ) [virtual]**

Specifies the command to be scheduled while the button is pressed The command will be scheduled repeatedly while the button is pressed and will be canceled when the button is released.

### Parameters:

**command** The pointer to the command to run

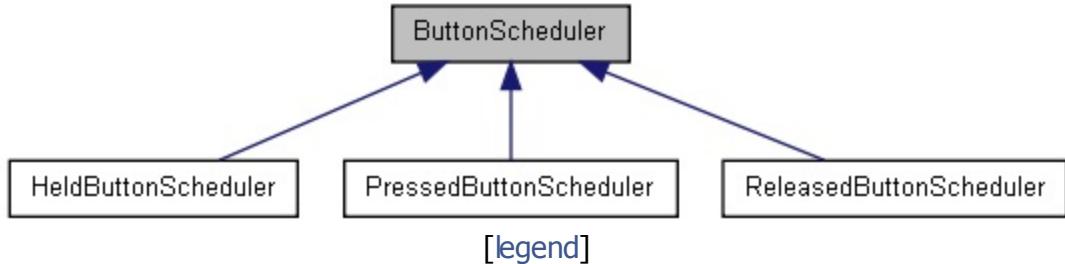
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/Buttons/**Button.h**
- C:/WindRiver/workspace/WPILib/Buttons/Button.cpp

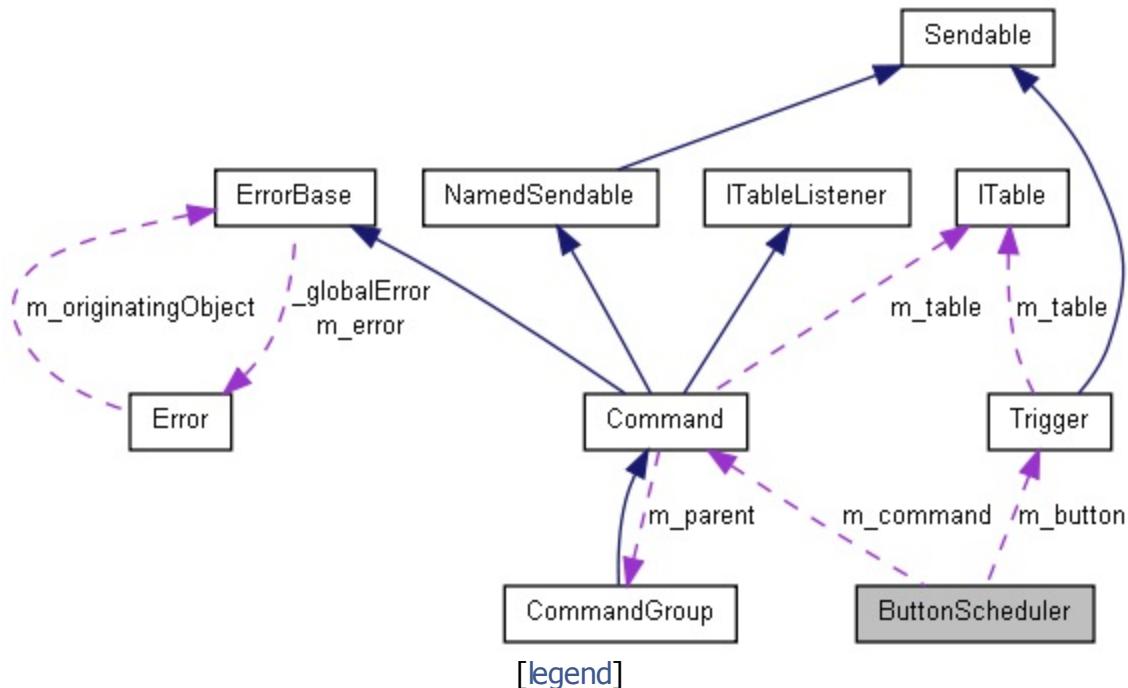


# ButtonScheduler Class Reference

## Inheritance diagram for ButtonScheduler:



## Collaboration diagram for ButtonScheduler:



## List of all members.

# Public Member Functions

```
    ButtonScheduler (bool last, Trigger *button, Command *orders)  
virtual void Execute ()=0  
void Start ()
```

    bool **m\_pressedLast**

**Trigger** \* **m\_button**

**Command** \* **m\_command**

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

- C:/WindRiver/workspace/WPILib/Buttons/**ButtonScheduler.h**
- C:/WindRiver/workspace/WPILib/Buttons/ButtonScheduler.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

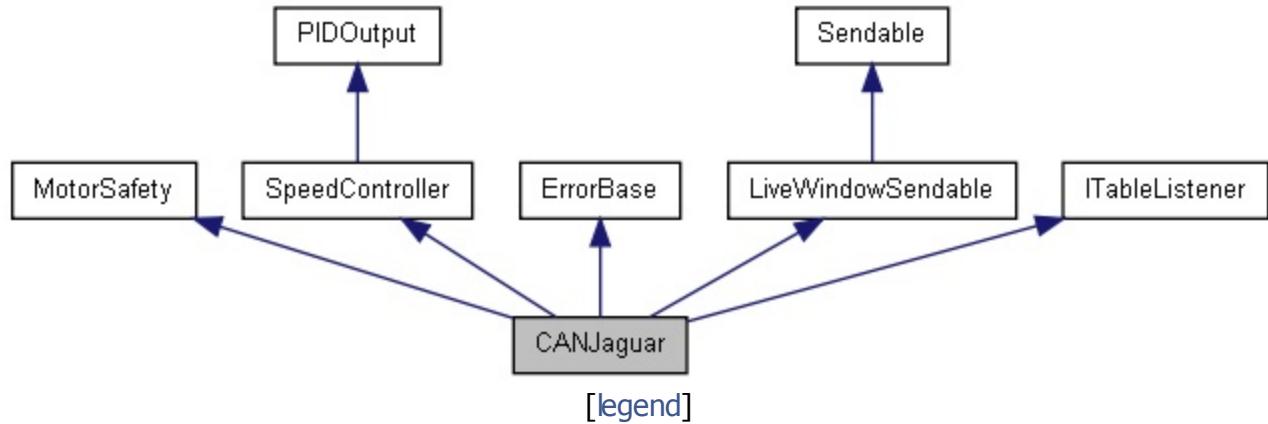
[Public Types](#) | [Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Static Public Attributes](#) |  
[Protected Member Functions](#) |  
[Static Protected Member Functions](#) |  
[Protected Attributes](#)

# CANJaguar Class Reference

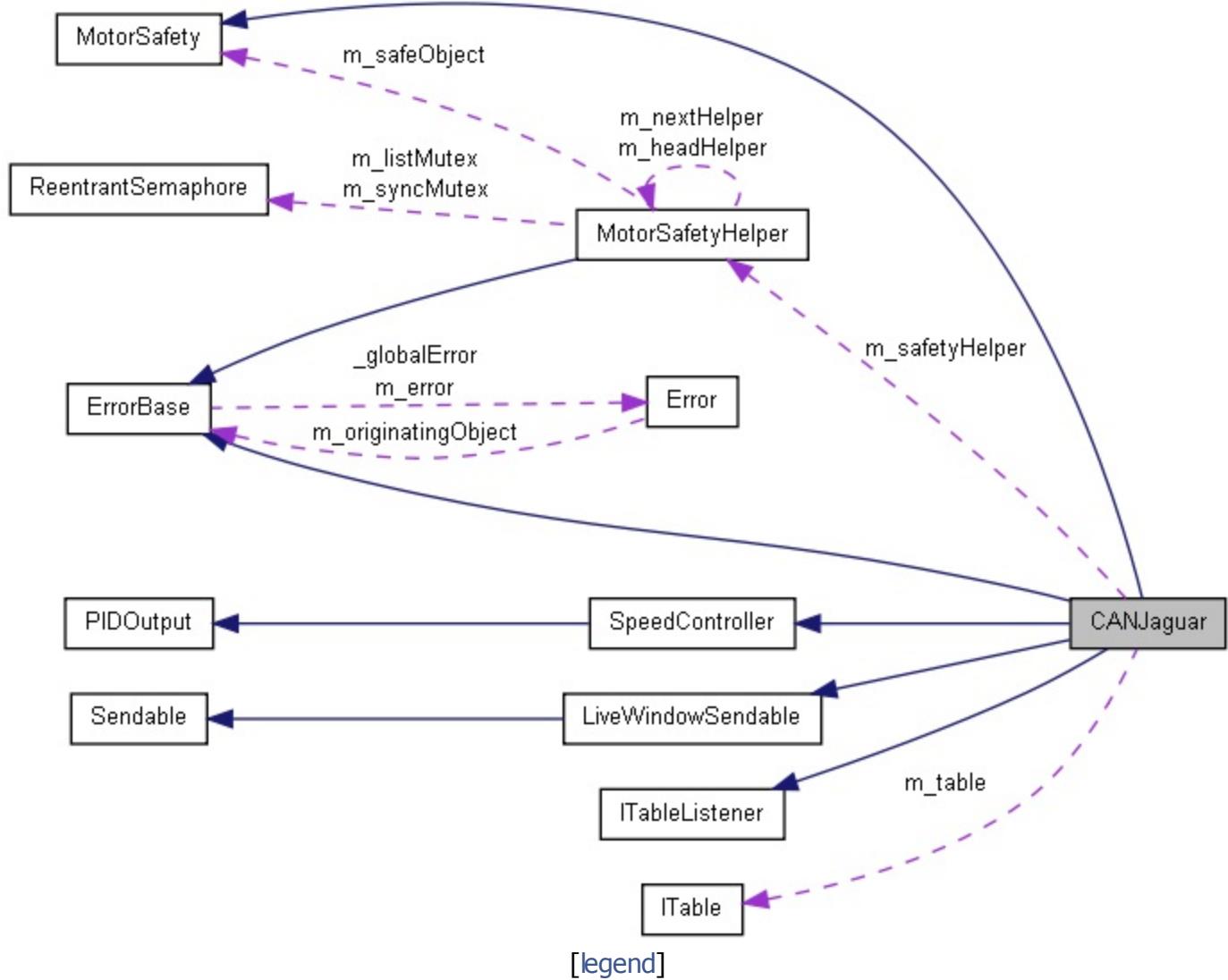
Luminary Micro **Jaguar** Speed Control. More...

```
#include <CANJaguar.h>
```

Inheritance diagram for CANJaguar:



Collaboration diagram for CANJaguar:



List of all members.

```
enum ControlMode {
    kPercentVbus, kCurrent, kSpeed, kPosition,
    kVoltage
}
enum Faults { kCurrentFault = 1, kTemperatureFault = 2,
    kBusVoltageFault = 4, kGateDriverFault = 8 }
enum Limits { kForwardLimit = 1, kReverseLimit = 2 }
enum PositionReference { kPosRef_QuadEncoder = 0,
    kPosRef_Potentiometer = 1, kPosRef_None = 0xFF }
enum SpeedReference { kSpeedRef_Encoder = 0,
    kSpeedRef_InvEncoder = 2, kSpeedRef_QuadEncoder
    = 3, kSpeedRef_None = 0xFF }
enum NeutralMode { kNeutralMode_Jumper = 0,
    kNeutralMode_Brake = 1, kNeutralMode_Coast = 2 }
enum LimitMode { kLimitMode_SwitchInputsOnly = 0,
    kLimitMode_SoftPositionLimits = 1 }
```

# Public Member Functions

**CANJaguar** (UINT8 deviceNumber, ControlMode controlMode=kPercentVbus)  
Constructor.

virtual float **Get ()**  
Get the recently set outputValue setpoint.

virtual void **Set (float value, UINT8 syncGroup=0)**  
Set the output set-point value.

virtual void **Disable ()**  
Common interface for disabling a motor.

virtual void **PIDWrite (float output)**  
Write out the PID value as seen in the **PIDOutput** base object.

void **SetSpeedReference (SpeedReference reference)**  
Set the reference source device for speed controller mode.

SpeedReference **GetSpeedReference ()**  
Get the reference source device for speed controller mode.

void **SetPositionReference (PositionReference reference)**  
Set the reference source device for position controller mode.

PositionReference **GetPositionReference ()**  
Get the reference source device for position controller mode.

void **SetPID (double p, double i, double d)**  
Set the P, I, and D constants for the closed loop modes.

double **GetP ()**  
Get the Proportional gain of the controller.

double **GetI ()**  
Get the Integral gain of the controller.

double **GetD ()**  
Get the Differential gain of the controller.

void **EnableControl (double encoderInitialPosition=0.0)**  
Enable the closed loop controller.

void **DisableControl ()**  
Disable the closed loop controller.

void **ChangeControlMode (ControlMode controlMode)**  
Change the control mode of this **Jaguar** object.

ControlMode **GetControlMode ()**  
Get the active control mode from the **Jaguar**.

float	<b>GetBusVoltage ()</b>	Get the voltage at the battery input terminals of the <b>Jaguar</b> .
float	<b>GetOutputVoltage ()</b>	Get the voltage being output from the motor terminals of the <b>Jaguar</b> .
float	<b>GetOutputCurrent ()</b>	Get the current through the motor terminals of the <b>Jaguar</b> .
float	<b>GetTemperature ()</b>	Get the internal temperature of the <b>Jaguar</b> .
double	<b>GetPosition ()</b>	Get the position of the encoder or potentiometer.
double	<b>GetSpeed ()</b>	Get the speed of the encoder.
bool	<b>GetForwardLimitOK ()</b>	Get the status of the forward limit switch.
bool	<b>GetReverseLimitOK ()</b>	Get the status of the reverse limit switch.
UINT16	<b>GetFaults ()</b>	Get the status of any faults the <b>Jaguar</b> has detected.
bool	<b>GetPowerCycled ()</b>	Check if the Jaguar's power has been cycled since this was last called.
void	<b>SetVoltageRampRate</b> (double rampRate)	Set the maximum voltage change rate.
virtual UINT32	<b>GetFirmwareVersion ()</b>	Get the version of the firmware running on the <b>Jaguar</b> .
UINT8	<b>GetHardwareVersion ()</b>	Get the version of the <b>Jaguar</b> hardware.
void	<b>ConfigNeutralMode</b> (NeutralMode mode)	Configure what the controller does to the H-Bridge when neutral (not driving the output).
void	<b>ConfigEncoderCodesPerRev</b> (UINT16 codesPerRev)	Configure how many codes per revolution are generated by your encoder.
void	<b>ConfigPotentiometerTurns</b> (UINT16 turns)	Configure the number of turns on the potentiometer.
void	<b>ConfigSoftPositionLimits</b> (double forwardLimitPosition, double reverseLimitPosition)	Configure Soft Position Limits when in Position <b>Controller</b>

mode.

void **DisableSoftPositionLimits ()**

Disable Soft Position Limits if previously enabled.

void **ConfigMaxOutputVoltage (double voltage)**

Configure the maximum voltage that the **Jaguar** will ever output.

void **ConfigFaultTime (float faultTime)**

Configure how long the **Jaguar** waits in the case of a fault before resuming operation.

void **SetExpiration (float timeout)**

float **GetExpiration ()**

bool **IsAlive ()**

void **StopMotor ()**

Common interface for stopping the motor Part of the **MotorSafety** interface.

bool **IsSafetyEnabled ()**

void **SetSafetyEnabled (bool enabled)**

void **GetDescription (char \*desc)**

static void **UpdateSyncGroup (UINT8 syncGroup)**

Update all the motors that have pending sets in the syncGroup.

## Static Public Attributes

```
static const INT32 kControllerRate = 1000  
static const double kApproxBusVoltage = 12.0
```

# Protected Member Functions

UINT8	<b>packPercentage</b> (UINT8 *buffer, double value)
UINT8	<b>packFXP8_8</b> (UINT8 *buffer, double value)
UINT8	<b>packFXP16_16</b> (UINT8 *buffer, double value)
UINT8	<b>packINT16</b> (UINT8 *buffer, INT16 value)
UINT8	<b>packINT32</b> (UINT8 *buffer, INT32 value)
double	<b>unpackPercentage</b> (UINT8 *buffer)
double	<b>unpackFXP8_8</b> (UINT8 *buffer)
double	<b>unpackFXP16_16</b> (UINT8 *buffer)
INT16	<b>unpackINT16</b> (UINT8 *buffer)
INT32	<b>unpackINT32</b> (UINT8 *buffer)
<b>setTransaction</b> (UINT32 messageID, const UINT8 *data,	

virtual void	<b>UINT8 dataSize)</b>	Execute a transaction with a <b>Jaguar</b> that sets some property.
virtual void	<b>getTransaction</b> (UINT32 messageID, UINT8 *data, UINT8 *dataSize)	Execute a transaction with a <b>Jaguar</b> that gets some property.
void	<b>ValueChanged</b> ( <b>ITable</b> *source, const std::string &key, <b>EntryValue</b> value, bool isNew)	Called when a key-value pair is changed in a <b>ITable</b> WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.
void	<b>UpdateTable</b> ()	Update the table for this sendable object with the latest values.
void	<b>StartLiveWindowMode</b> ()	Start having this sendable object automatically respond to value changes reflect the value on the table.
void	<b>StopLiveWindowMode</b> ()	Stop having this sendable object automatically respond to value changes.
std::string	<b>GetSmartDashboardType</b> ()	
void	<b>InitTable</b> ( <b>ITable</b> *subTable)	Initializes a table for this sendable object.
<b>ITable</b> *	<b>GetTable</b> ()	

# Static Protected Member Functions

static INT32	<b>sendMessage</b> (UINT32 messageID, const UINT8 *data, UINT8 dataSize) Send a message on the CAN bus through the CAN driver in FRC_NetworkCommunication.
static INT32	<b>receiveMessage</b> (UINT32 *messageID, UINT8 *data, UINT8 *dataSize, float timeout=0.02) Receive a message from the CAN bus through the CAN driver in FRC_NetworkCommunication.

# Protected Attributes

UINT8	<b>m_deviceNumber</b>
ControlMode	<b>m_controlMode</b>
SEM_ID	<b>m_transactionSemaphore</b>
double	<b>m_maxOutputVoltage</b>
<b>MotorSafetyHelper</b> *	<b>m_safetyHelper</b>
<b>ITable</b> *	<b>m_table</b>

# Detailed Description

Luminary Micro **Jaguar** Speed Control.

---

# Constructor & Destructor Documentation

```
CANJaguar::CANJaguar( UINT8 deviceNumber,
                      ControlMode controlMode = kPercentVbus [
                      ) ] explicit
```

Constructor.

## Parameters:

**deviceNumber** The the address of the **Jaguar** on the CAN bus.

# Member Function Documentation

## **void CANJaguar::ChangeControlMode ( ControlMode controlMode )**

Change the control mode of this **Jaguar** object.

After changing modes, configure any PID constants or other settings needed and then **EnableControl()** to actually change the mode on the **Jaguar**.

### **Parameters:**

**controlMode** The new mode.

## **void CANJaguar::ConfigEncoderCodesPerRev ( UINT16 codesPerRev )**

Configure how many codes per revolution are generated by your encoder.

### **Parameters:**

**codesPerRev** The number of counts per revolution in 1X mode.

## **void CANJaguar::ConfigFaultTime ( float faultTime )**

Configure how long the **Jaguar** waits in the case of a fault before resuming operation.

Faults include over temperature, over current, and bus under voltage. The default is 3.0 seconds, but can be reduced to as low as 0.5 seconds.

### **Parameters:**

**faultTime** The time to wait before resuming operation, in seconds.

## **void CANJaguar::ConfigMaxOutputVoltage ( double voltage )**

Configure the maximum voltage that the **Jaguar** will ever output.

This can be used to limit the maximum output voltage in all modes so that motors which cannot withstand full bus voltage can be used safely.

### **Parameters:**

**voltage** The maximum voltage output by the **Jaguar**.

## **void CANJaguar::ConfigNeutralMode ( NeutralMode mode )**

Configure what the controller does to the H-Bridge when neutral (not driving the output).

This allows you to override the jumper configuration for brake or coast.

### **Parameters:**

**mode** Select to use the jumper setting or to override it to coast or brake.

## **void CANJaguar::ConfigPotentiometerTurns ( UINT16 turns )**

Configure the number of turns on the potentiometer.

There is no special support for continuous turn potentiometers. Only integer numbers of turns are supported.

### **Parameters:**

**turns** The number of turns of the potentiometer

## **void CANJaguar::ConfigSoftPositionLimits ( double forwardLimitPosition, double reverseLimitPosition )**

Configure Soft Position Limits when in Position **Controller** mode.

When controlling position, you can add additional limits on top of the limit switch inputs that are based on the position feedback. If the position limit is reached or the switch is opened, that direction will be disabled.

### **Parameters:**

**forwardLimitPosition** The position that if exceeded will disable the forward direction.

**reverseLimitPosition** The position that if exceeded will disable the reverse direction.

## **void CANJaguar::Disable ( ) [virtual]**

Common interface for disabling a motor.

## **Deprecated:**

Call DisableControl instead.

Implements **SpeedController**.

### **void CANJaguar::DisableControl( )**

Disable the closed loop controller.

Stop driving the output based on the feedback.

### **void CANJaguar::DisableSoftPositionLimits( )**

Disable Soft Position Limits if previously enabled.

Soft Position Limits are disabled by default.

### **void CANJaguar::EnableControl( double encoderInitialPosition = 0.0 )**

Enable the closed loop controller.

Start actually controlling the output based on the feedback. If starting a position controller with an encoder reference, use the encoderInitialPosition parameter to initialize the encoder state.

#### **Parameters:**

**encoderInitialPosition** **Encoder** position to set if position with encoder reference. Ignored otherwise.

### **float CANJaguar::Get( ) [virtual]**

Get the recently set outputValue setpoint.

The scale and the units depend on the mode the **Jaguar** is in. In PercentVbus Mode, the outputValue is from -1.0 to 1.0 (same as **PWM Jaguar**). In Voltage Mode, the outputValue is in Volts. In Current Mode, the outputValue is in Amps. In Speed Mode, the outputValue is in Rotations/Minute. In Position Mode, the outputValue is in Rotations.

#### **Returns:**

The most recently set outputValue setpoint.

Implements **SpeedController**.

### **float CANJaguar::GetBusVoltage( )**

Get the voltage at the battery input terminals of the **Jaguar**.

**Returns:**

The bus voltage in Volts.

### **CANJaguar::ControlMode CANJaguar::GetControlMode( )**

Get the active control mode from the **Jaguar**.

Ask the Jag what mode it is in.

**Returns:**

ControlMode that the Jag is in.

### **double CANJaguar::GetD( )**

Get the Differential gain of the controller.

**Returns:**

The differential gain.

### **UINT16 CANJaguar::GetFaults( )**

Get the status of any faults the **Jaguar** has detected.

**Returns:**

A bit-mask of faults defined by the "Faults" enum.

### **UINT32 CANJaguar::GetFirmwareVersion( ) [virtual]**

Get the version of the firmware running on the **Jaguar**.

**Returns:**

The firmware version. 0 if the device did not respond.

## **bool CANJaguar::GetForwardLimitOK( )**

Get the status of the forward limit switch.

### **Returns:**

The motor is allowed to turn in the forward direction when true.

## **UINT8 CANJaguar::GetHardwareVersion( )**

Get the version of the **Jaguar** hardware.

### **Returns:**

The hardware version. 1: **Jaguar**, 2: Black **Jaguar**

## **double CANJaguar::GetI( )**

Get the Intregal gain of the controller.

### **Returns:**

The integral gain.

## **float CANJaguar::GetOutputCurrent( )**

Get the current through the motor terminals of the **Jaguar**.

### **Returns:**

The output current in Amps.

## **float CANJaguar::GetOutputVoltage( )**

Get the voltage being output from the motor terminals of the **Jaguar**.

### **Returns:**

The output voltage in Volts.

## **double CANJaguar::GetP( )**

Get the Proportional gain of the controller.

**Returns:**

The proportional gain.

**double CANJaguar::GetPosition( )**

Get the position of the encoder or potentiometer.

**Returns:**

The position of the motor in rotations based on the configured feedback.

**CANJaguar::PositionReference CANJaguar::GetPositionReference( )**

Get the reference source device for position controller mode.

**Returns:**

A PositionReference indicating the currently selected reference device for position controller mode.

**bool CANJaguar::GetPowerCycled( )**

Check if the Jaguar's power has been cycled since this was last called.

This should return true the first time called after a **Jaguar** power up, and false after that.

**Returns:**

The **Jaguar** was power cycled since the last call to this function.

**bool CANJaguar::GetReverseLimitOK( )**

Get the status of the reverse limit switch.

**Returns:**

The motor is allowed to turn in the reverse direction when true.

**std::string CANJaguar::GetSmartDashboardType( )** [protected, virtual]

**Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

**double CANJaguar::GetSpeed( )**

Get the speed of the encoder.

**Returns:**

The speed of the motor in RPM based on the configured feedback.

**CANJaguar::SpeedReference CANJaguar::GetSpeedReference( )**

Get the reference source device for speed controller mode.

**Returns:**

A SpeedReference indicating the currently selected reference device for speed controller mode.

**ITable \* CANJaguar::GetTable( ) [protected, virtual]****Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

**float CANJaguar::GetTemperature( )**

Get the internal temperature of the **Jaguar**.

**Returns:**

The temperature of the **Jaguar** in degrees Celsius.

**void CANJaguar::getTransaction( UINT32 messageID,  
                              UINT8 \* data,**

**UINT8 \* dataSize** [protected, virtual]  
)

Execute a transaction with a **Jaguar** that gets some property.

**Jaguar** always generates a message with the same message ID when replying.

## Parameters:

- messageID** The messageID to read from the CAN bus (device number is added internally)
- data** The up to 8 bytes of data that was received with the message
- dataSize** Indicates how much data was received

**void CANJaguar::InitTable( ITable \* subtable )** [protected, virtual]

Initializes a table for this sendable object.

## Parameters:

**subtable** The table to put the values in.

## Implements **Sendable**.

**void CANJaguar::PIDWrite( float output ) [virtual]**

Write out the PID value as seen in the **PIDOutput** base object.

## Deprecated:

Call Set instead.

## Parameters:

**output** Write out the PercentVbus value as was computed by the **PIDController**

Implements **PIDOutput**.

Receive a message from the CAN bus through the CAN driver in FRC\_NetworkCommunication.

### Parameters:

- messageID** The messageID to read from the CAN bus
- data** The up to 8 bytes of data that was received with the message
- dataSize** Indicates how much data was received
- timeout** Specify how long to wait for a message (in seconds)

### Returns:

Status of receive call

```
INT32 CANJaguar::sendMessage ( UINT32 messageID,
                                const UINT8 * data,
                                UINT8 dataSize
                                ) [static, protected]
```

Send a message on the CAN bus through the CAN driver in FRC\_NetworkCommunication.

Trusted messages require a 2-byte token at the beginning of the data payload. If the message being sent is trusted, make space for the token.

### Parameters:

- messageID** The messageID to be used on the CAN bus
- data** The up to 8 bytes of data to be sent with the message
- dataSize** Specify how much of the data in "data" to send

### Returns:

Status of send call

```
void CANJaguar::Set ( float outputValue,
                      UINT8 syncGroup = 0
                      ) [virtual]
```

Set the output set-point value.

The scale and the units depend on the mode the **Jaguar** is in. In PercentVbus Mode, the outputValue is from -1.0 to 1.0 (same as **PWM Jaguar**). In Voltage Mode, the outputValue is in Volts. In Current Mode, the outputValue is in Amps. In Speed Mode, the outputValue is in Rotations/Minute. In Position Mode, the outputValue is in Rotations.

### Parameters:

- outputValue** The set-point to sent to the motor controller.  
**syncGroup** The update group to add this **Set()** to, pending **UpdateSyncGroup()**. If 0, update immediately.

Implements **SpeedController**.

```
void CANJaguar::SetPID( double p,  
                        double i,  
                        double d  
)
```

Set the P, I, and D constants for the closed loop modes.

### Parameters:

- p** The proportional gain of the Jaguar's PID controller.  
**i** The integral gain of the Jaguar's PID controller.  
**d** The differential gain of the Jaguar's PID controller.

```
void CANJaguar::SetPositionReference( PositionReference reference )
```

Set the reference source device for position controller mode.

Choose between using an encoder and using a potentiometer as the source of position feedback when in position control mode.

### Parameters:

- reference** Specify a PositionReference.

```
void CANJaguar::SetSpeedReference( SpeedReference reference )
```

Set the reference source device for speed controller mode.

Choose encoder as the source of speed feedback when in speed control mode.

## Parameters:

**reference** Specify a SpeedReference.

```
void CANJaguar::setTransaction ( UINT32 messageID,  
                                const UINT8 * data,  
                                UINT8 dataSize  
) [protected, virtual]
```

Execute a transaction with a **Jaguar** that sets some property.

**Jaguar** always acks when it receives a message. If we don't wait for an ack, the message object in the **Jaguar** could get overwritten before it is handled.

## Parameters:

**messageID** The messageID to be used on the CAN bus (device number is added internally)

**data** The up to 8 bytes of data to be sent with the message

**dataSize** Specify how much of the data in "data" to send

```
void CANJaguar::SetVoltageRampRate ( double rampRate )
```

Set the maximum voltage change rate.

When in PercentVbus or Voltage output mode, the rate at which the voltage changes can be limited to reduce current spikes. Set this to 0.0 to disable rate limiting.

## Parameters:

**rampRate** The maximum rate of voltage change in Percent Voltage mode in V/s.

```
void CANJaguar::StopMotor ( ) [virtual]
```

Common interface for stopping the motor Part of the **MotorSafety** interface.

## Deprecated:

Call DisableControl instead.

Implements **MotorSafety**.

## **void CANJaguar::UpdateSyncGroup ( **UINT8 syncGroup** ) [static]**

Update all the motors that have pending sets in the syncGroup.

### **Parameters:**

**syncGroup** A bitmask of groups to generate synchronous output.

## **void CANJaguar::ValueChanged ( **ITable \* source,**                                  **const std::string & key,**                                  **EntryValue value,**                                  **bool isNew** ) [protected, virtual]**

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

### **Parameters:**

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

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

- C:/WindRiver/workspace/WPILib/**CANJaguar.h**
- C:/WindRiver/workspace/WPILib/CANJaguar.cpp

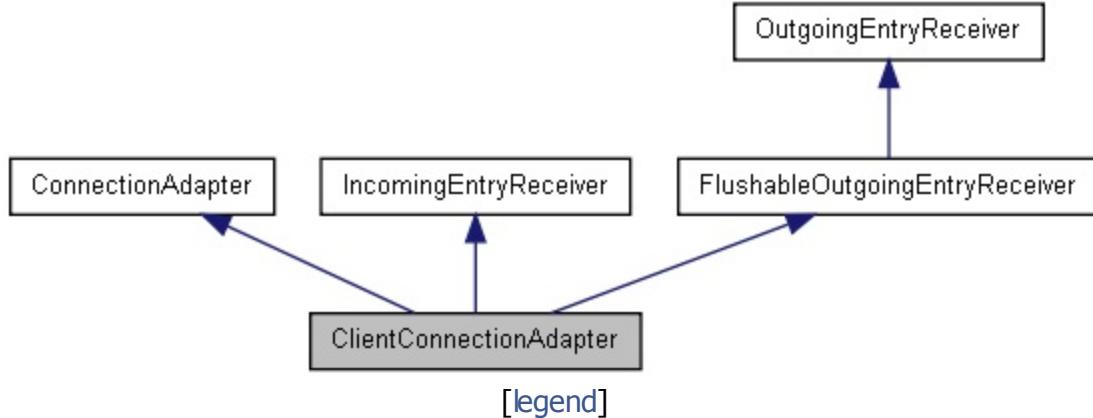


# **ClientConnectionAdapter Class Reference**

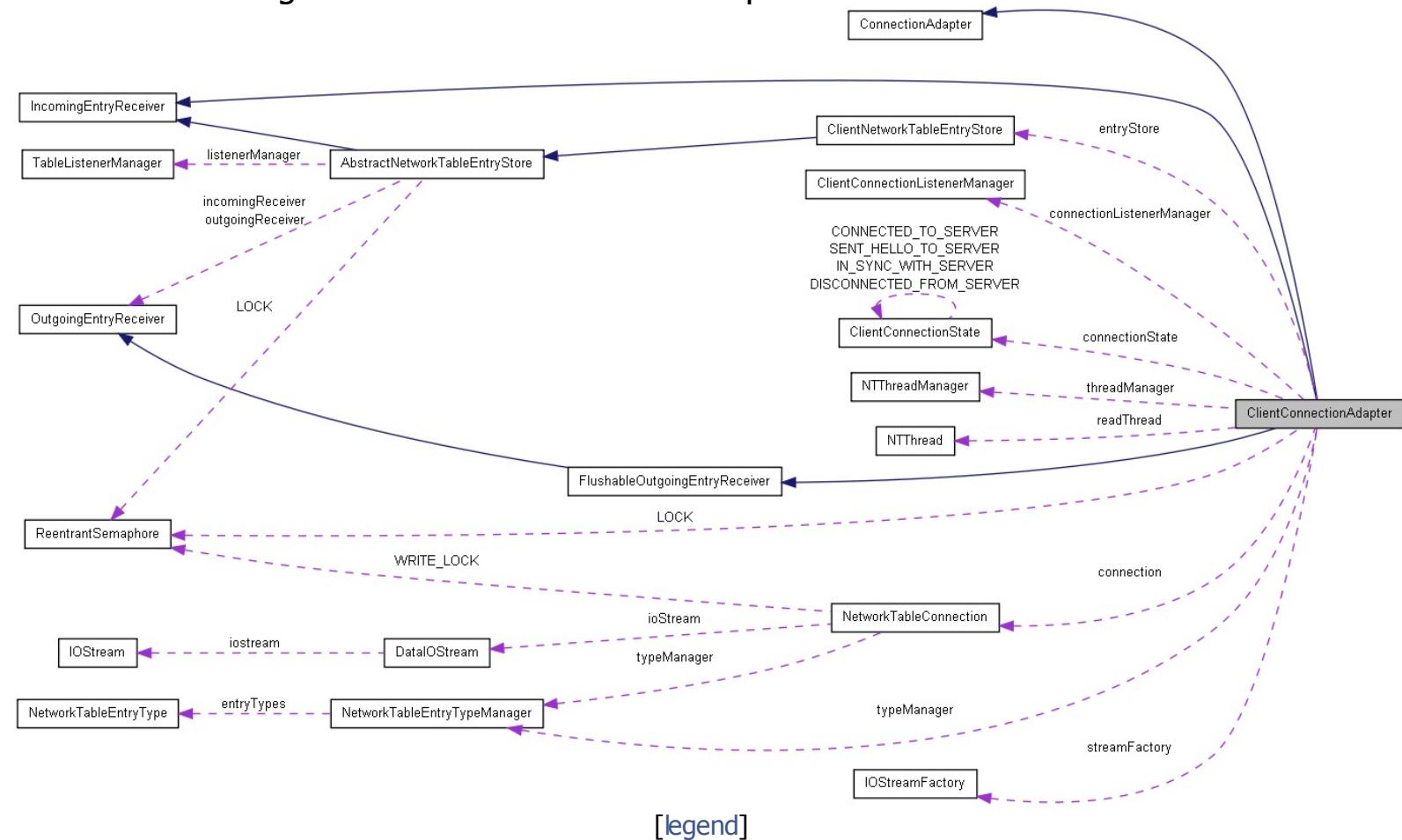
Object that adapts messages from a server. [More...](#)

```
#include <ClientConnectionAdapter.h>
```

## Inheritance diagram for ClientConnectionAdapter:



## Collaboration diagram for ClientConnectionAdapter:



## List of all members.

# Public Member Functions

<b>ClientConnectionState *</b>	<b>getConnectionState ()</b>
bool	<b>isConnected ()</b> <b>ClientConnectionAdapter</b> ( <b>ClientNetworkTableEntryStore</b> &entryStore, <b>NTThreadManager</b> &threadManager, <b>IOStreamFactory</b> &streamFactory, <b>ClientConnectionListenerManager</b> &connectionListenerManager, <b>NetworkTableEntryTypeManager</b> &typeManager) Create a new <b>ClientConnectionAdapter</b> .
void	<b>reconnect ()</b> Reconnect the client to the server (even if the client is not currently connected)
void	<b>close ()</b> Close the client connection.
void	<b>close (ClientConnectionState *newState)</b> Close the connection to the server and enter the given state.
void	<b>badMessage (BadMessageException &amp;e)</b> called if a bad message exception is thrown
void	<b>ioException (IOException &amp;e)</b> called if an io exception is thrown
<b>NetworkTableEntry *</b>	<b>GetEntry (EntryId id)</b>
void	<b>keepAlive ()</b>
void	<b>clientHello (ProtocolVersion protocolRevision)</b>
void	<b>protocolVersionUnsupported (ProtocolVersion protocolRevision)</b>
void	<b>serverHelloComplete ()</b>
void	<b>offerIncomingAssignment (NetworkTableEntry *entry)</b>
void	<b>offerIncomingUpdate (NetworkTableEntry *entry, SequenceNumber sequenceNumber, EntryValue value)</b>
void	<b>offerOutgoingAssignment (NetworkTableEntry *entry)</b>
void	<b>offerOutgoingUpdate (NetworkTableEntry *entry)</b>

void **flush** ()  
void **ensureAlive** ()

---

# **Detailed Description**

Object that adapts messages from a server.

## **Author:**

Mitchell

---

# Constructor & Destructor Documentation

**ClientConnectionAdapter::ClientConnectionAdapter(** ClientNetworkTableEntry  
NTThreadManager &  
IOStreamFactory &  
ClientConnectionListener  
NetworkTableEntryType1  
**)**

Create a new **ClientConnectionAdapter**.

## Parameters:

**entryStore**  
**threadManager**  
**streamFactory**  
**transactionPool**  
**connectionListenerManager**

# Member Function Documentation

**void ClientConnectionAdapter::badMessage ( BadMessageException & e ) [virtual]**

called if a bad message exception is thrown

## Parameters:

e

Implements **ConnectionAdapter**.

**void ClientConnectionAdapter::close ( ClientConnectionState \* newState )**

Close the connection to the server and enter the given state.

## Parameters:

newState

**ClientConnectionState \* ClientConnectionAdapter::getConnectionState ( )**

## Returns:

the state of the connection

**void ClientConnectionAdapter::ioException ( IOException & e ) [virtual]**

called if an io exception is thrown

## Parameters:

e

Implements **ConnectionAdapter**.

**bool ClientConnectionAdapter::isConnected ( )**

## Returns:

if the client is connected to the server

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

- C:/WindRiver/workspace/WPILib/networktables2/client/[ClientConnectionAdapter.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/client/ClientConnectionAdapter.cpp

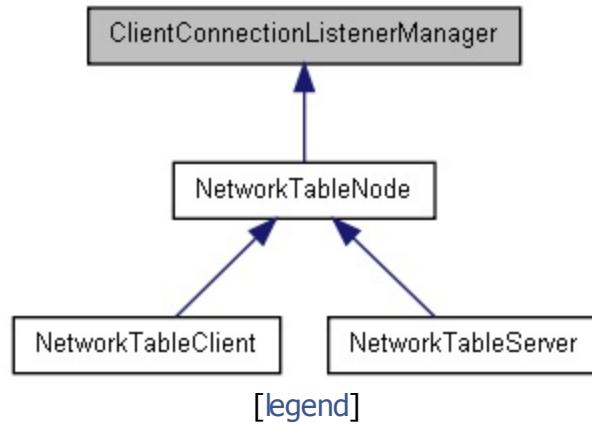


# ClientConnectionListenerManager Class Reference

An object that manages connection listeners and fires events for other listeners. More...

```
#include <ClientConnectionListenerManager.h>
```

Inheritance diagram for ClientConnectionListenerManager:



List of all members.

## Public Member Functions

virtual void **FireConnectedEvent ()=0**  
called when something is connected

virtual void **FireDisconnectedEvent ()=0**  
called when something is disconnected

---

# Detailed Description

An object that manages connection listeners and fires events for other listeners.

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/networktables2/client/[ClientConnectionListener.h](#)

---

Generated by  1.7.2

[Class List](#)[Class Hierarchy](#)[Class Members](#)

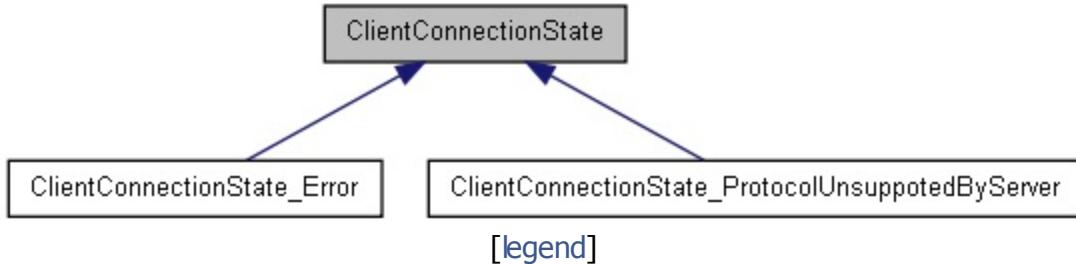
[Public Member Functions](#) |  
[Static Public Attributes](#) |  
[Protected Member Functions](#)

# ClientConnectionState Class Reference

Represents a state that the client is in. [More...](#)

```
#include <ClientConnectionState.h>
```

Inheritance diagram for ClientConnectionState:



[legend]

Collaboration diagram for ClientConnectionState:



[legend]

List of all members.

# Public Member Functions

virtual const char \* **toString ()**

static <b>ClientConnectionState</b>	<b>DISCONNECTED_FROM_SERVER</b>	indicates that the client is disconnected from the server
static <b>ClientConnectionState</b>	<b>CONNECTED_TO_SERVER</b>	indicates that the client is connected to the server but has not yet begun communication
static <b>ClientConnectionState</b>	<b>SENT_HELLO_TO_SERVER</b>	represents that the client has sent the hello to the server and is waiting for a response
static <b>ClientConnectionState</b>	<b>IN_SYNC_WITH_SERVER</b>	represents that the client is now in sync with the server



# Detailed Description

Represents a state that the client is in.

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/networktables2/client/[ClientConnectionState.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/client/ClientConnectionState.cpp

---

Generated by  1.7.2

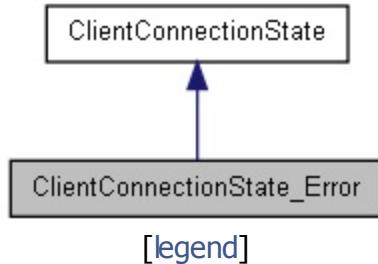


# ClientConnectionState\_Error Class Reference

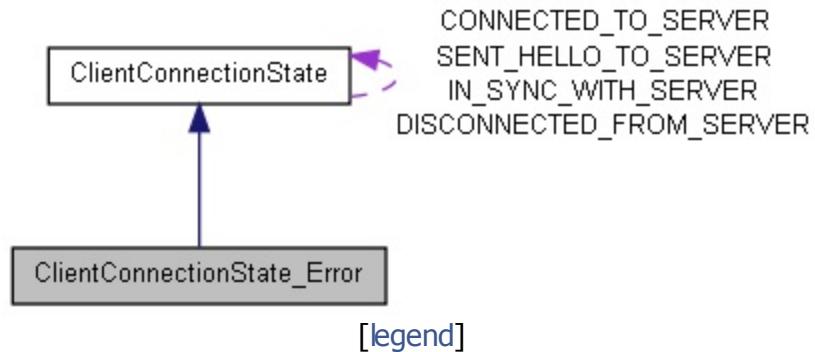
Represents that the client is in an error state. [More...](#)

```
#include <ClientConnectionState.h>
```

Inheritance diagram for ClientConnectionState\_Error:



Collaboration diagram for ClientConnectionState\_Error:



[List of all members.](#)

## Public Member Functions

**ClientConnectionState\_Error** (std::exception &e)

Create a new error state.

std::exception & **getException** ()

virtual const char \* **toString** ()

---

## Detailed Description

Represents that the client is in an error state.

### **Author:**

Mitchell

---

# Constructor & Destructor Documentation

## **ClientConnectionState\_Error::ClientConnectionState\_Error ( std::exception &**

Create a new error state.

### **Parameters:**

**e**

# Member Function Documentation

## **std::exception & ClientConnectionState\_Error::getException( )**

### Returns:

the exception that caused the client to enter an error state

---

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

- C:/WindRiver/workspace/WPILib/networktables2/client/[ClientConnectionState.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/client/ClientConnectionState.cpp

Generated by  1.7.2

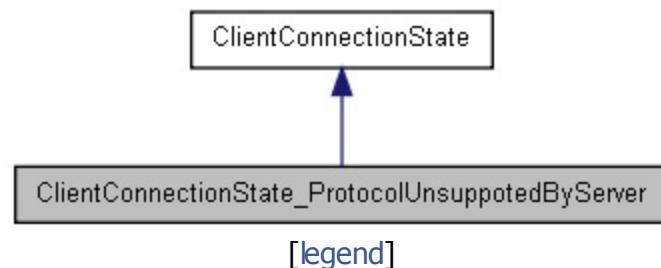


# ClientConnectionState\_ProtocolUnsupportedByServer Class Reference

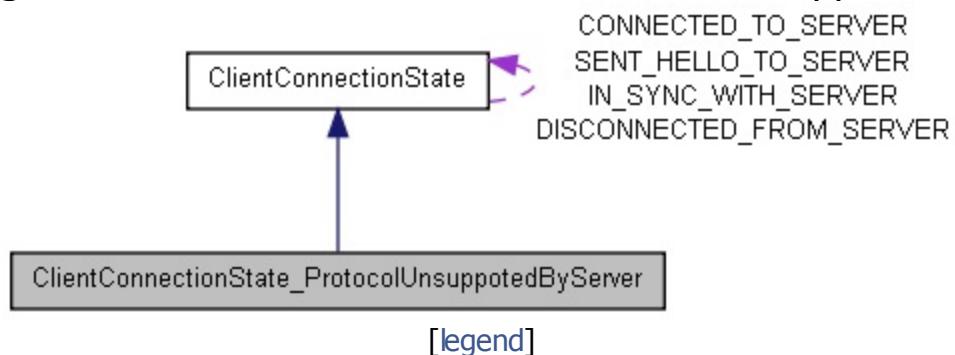
Represents that a client received a message from the server indicating that the client's protocol revision is not supported by the server. [More...](#)

```
#include <ClientConnectionState.h>
```

Inheritance diagram for ClientConnectionState\_ProtocolUnsupportedByServer:



Collaboration diagram for ClientConnectionState\_ProtocolUnsupportedByServer:



[List of all members.](#)

## Public Member Functions

### **ClientConnectionState\_ProtocolUnsupportedByServer**

(ProtocolVersion serverVersion)

Create a new protocol unsupported state.

ProtocolVersion **getServerVersion ()**

const char \* **toString ()**

## Detailed Description

Represents that a client received a message from the server indicating that the client's protocol revision is not supported by the server.

### Author:

Mitchell

---

# Constructor & Destructor Documentation

## **ClientConnectionState\_ProtocolUnsupportedByServer::ClientConnectionState\_**

Create a new protocol unsupported state.

### **Parameters:**

**serverVersion**

---

# Member Function Documentation

## ProtocolVersion ClientConnectionState::ProtocolUnsupportedByServer::getServerProtocolVersion()

### Returns:

the protocol version that the server reported it supports

---

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

- C:/WindRiver/workspace/WPILib/networktables2/client/[ClientConnectionState.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/client/ClientConnectionState.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

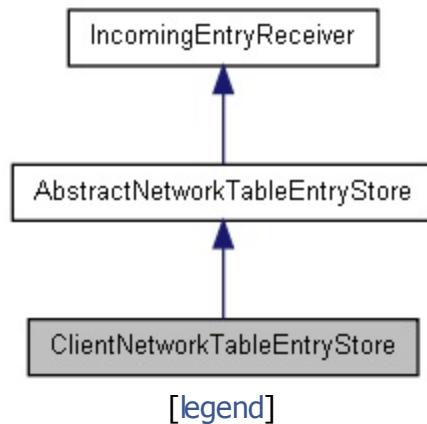
[Public Member Functions](#) |  
[Protected Member Functions](#)

# ClientNetworkTableEntryStore Class Reference

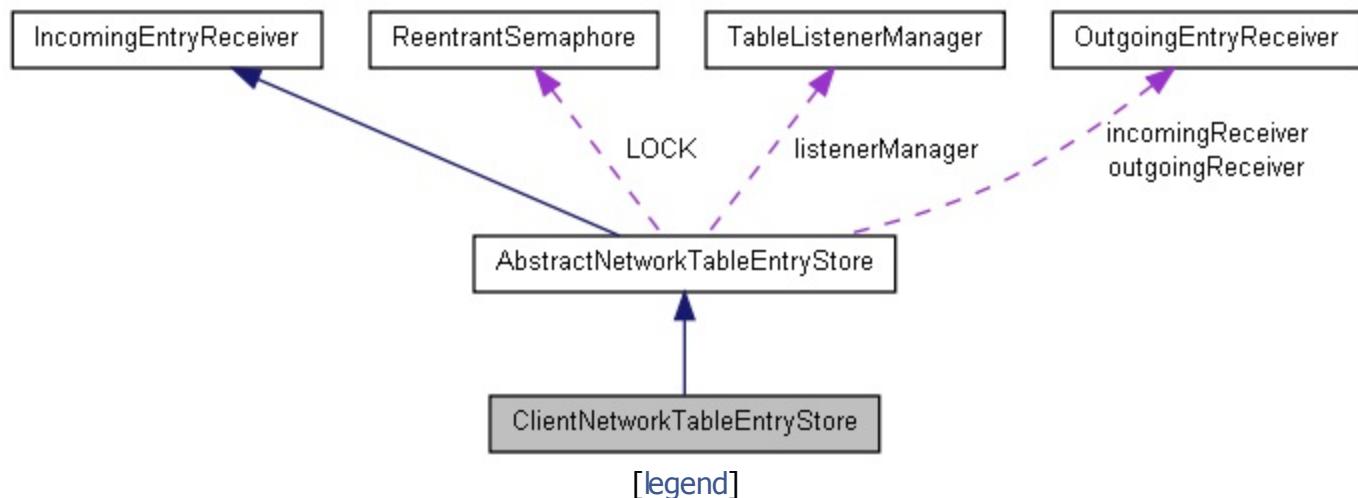
The entry store for a **NetworkTableClient**. More...

```
#include <ClientNetworkTableEntryStore.h>
```

Inheritance diagram for ClientNetworkTableEntryStore:



Collaboration diagram for ClientNetworkTableEntryStore:



List of all members.

# Public Member Functions

**ClientNetworkTableEntryStore** (**TableListenerManager** &listenerManager)  
Create a new **ClientNetworkTableEntryStore**.

void **sendUnknownEntries** (**NetworkTableConnection** &connection)  
Send all unknown entries in the entry store to the given connection.

---

bool **addEntry** (**NetworkTableEntry** \*newEntry)  
**updateEntry** (**NetworkTableEntry** \*entry, SequenceNumber  
bool sequenceNumber, **EntryValue** value)

---

# Detailed Description

The entry store for a [NetworkTableClient](#).

## Author:

Mitchell

---

# Constructor & Destructor Documentation

## **ClientNetworkTableEntryStore::ClientNetworkTableEntryStore ( TableListener<Table> & listenerManager )**

Create a new **ClientNetworkTableEntryStore**.

### Parameters:

**transactionPool**

**listenerManager**

# Member Function Documentation

## **void ClientNetworkTableEntryStore::sendUnknownEntries ( NetworkTableCon**

Send all unknown entries in the entry store to the given connection.

### **Parameters:**

**connection**

### **Exceptions:**

**IOException**

---

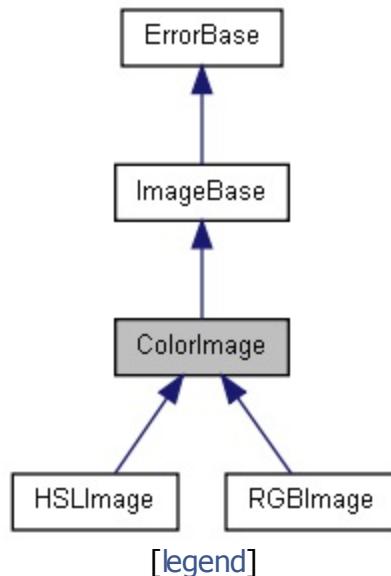
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/client/**ClientNetworkTableEntry**
- C:/WindRiver/workspace/WPILib/networktables2/client/ClientNetworkTableEntryStor

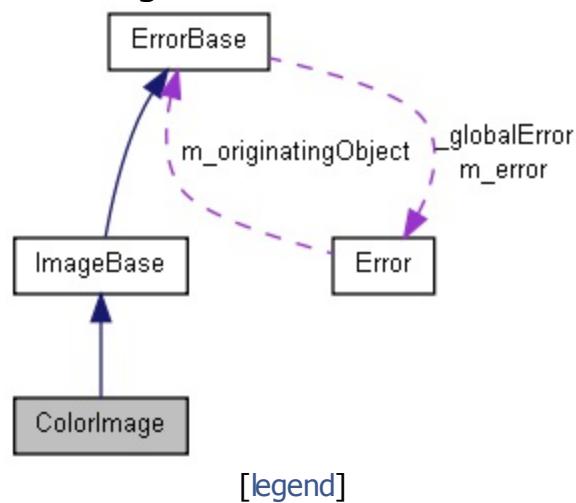


# ColorImage Class Reference

Inheritance diagram for ColorImage:



Collaboration diagram for ColorImage:



List of all members.

# Public Member Functions

	<b>ColorImage</b> ( <b>ImageType</b> type)
<b>BinaryImage</b> *	<b>ThresholdRGB</b> (int redLow, int redHigh, int greenLow, int greenHigh, int blueLow, int blueHigh) Perform a threshold in RGB space.
<b>BinaryImage</b> *	<b>ThresholdHSL</b> (int hueLow, int hueHigh, int saturationLow, int saturationHigh, int luminenceLow, int luminenceHigh) Perform a threshold in HSL space.
<b>BinaryImage</b> *	<b>ThresholdHSV</b> (int hueLow, int hueHigh, int saturationLow, int saturationHigh, int valueHigh, int valueLow) Perform a threshold in HSV space.
<b>BinaryImage</b> *	<b>ThresholdHSI</b> (int hueLow, int hueHigh, int saturationLow, int saturationHigh, int intensityLow, int intensityHigh) Perform a threshold in HSI space.
<b>BinaryImage</b> *	<b>ThresholdRGB</b> ( <b>Threshold</b> &threshold) Perform a threshold in RGB space.
<b>BinaryImage</b> *	<b>ThresholdHSL</b> ( <b>Threshold</b> &threshold) Perform a threshold in HSL space.
<b>BinaryImage</b> *	<b>ThresholdHSV</b> ( <b>Threshold</b> &threshold) Perform a threshold in HSV space.
<b>BinaryImage</b> *	<b>ThresholdHSI</b> ( <b>Threshold</b> &threshold) Perform a threshold in HSI space.
<b>MonoImage</b> *	<b>GetRedPlane</b> ()
<b>MonoImage</b> *	<b>GetGreenPlane</b> ()
<b>MonoImage</b> *	<b>GetBluePlane</b> ()
<b>MonoImage</b> *	<b>GetHSLHuePlane</b> ()
<b>MonoImage</b> *	<b>GetHSVHuePlane</b> ()
<b>MonoImage</b> *	<b>GetHSIHuePlane</b> ()
<b>MonoImage</b> *	<b>GetHSLSaturationPlane</b> ()
<b>MonoImage</b> *	<b>GetHVSaturationPlane</b> ()
<b>MonoImage</b> *	<b>GetHSISaturationPlane</b> ()
<b>MonoImage</b> *	<b>GetLuminancePlane</b> ()
<b>MonoImage</b> *	<b>GetValuePlane</b> ()
<b>MonoImage</b> *	<b>GetIntensityPlane</b> ()
void	<b>ReplaceRedPlane</b> ( <b>MonoImage</b> *plane) Replace the red color plane with a <b>MonoImage</b> .
void	<b>ReplaceGreenPlane</b> ( <b>MonoImage</b> *plane) Replace the green color plane with a <b>MonoImage</b> .

void	<b>ReplaceBluePlane (MonoImage *plane)</b> Replace the blue color plane with a <b>MonoImage</b> .
void	<b>ReplaceHSLHuePlane (MonoImage *plane)</b> Replace the Hue color plane in a HSL image with a <b>MonoImage</b> .
void	<b>ReplaceHSVHuePlane (MonoImage *plane)</b> Replace the Hue color plane in a HSV image with a <b>MonoImage</b> .
void	<b>ReplaceHSIHuePlane (MonoImage *plane)</b> Replace the first Hue plane in a HSI image with a <b>MonoImage</b> .
void	<b>ReplaceHSLSaturationPlane (MonoImage *plane)</b> Replace the Saturation color plane in an HSL image with a <b>MonoImage</b> .
void	<b>ReplaceHVSaturationPlane (MonoImage *plane)</b> Replace the Saturation color plane in a HSV image with a <b>MonoImage</b> .
void	<b>ReplaceHSISaturationPlane (MonoImage *plane)</b> Replace the Saturation color plane in a HSI image with a <b>MonoImage</b> .
void	<b>ReplaceLuminancePlane (MonoImage *plane)</b> Replace the Luminance color plane in an HSL image with a <b>MonoImage</b> .
void	<b>ReplaceValuePlane (MonoImage *plane)</b> Replace the Value color plane in an HSV with a <b>MonoImage</b> .
void	<b>ReplaceIntensityPlane (MonoImage *plane)</b> Replace the Intensity color plane in a HSI image with a <b>MonoImage</b> .
void	<b>ColorEqualize ()</b>
void	<b>LuminanceEqualize ()</b>

# Member Function Documentation

## **void ColorImage::ReplaceBluePlane ( MonoImage \* plane )**

Replace the blue color plane with a **MonoImage**.

### Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a **MonoImage** that will replace the specified color plane.

## **void ColorImage::ReplaceGreenPlane ( MonoImage \* plane )**

Replace the green color plane with a **MonoImage**.

### Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a **MonoImage** that will replace the specified color plane.

## **void ColorImage::ReplaceHSIHuePlane ( MonoImage \* plane )**

Replace the first Hue plane in a HSI image with a **MonoImage**.

### Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a **MonoImage** that will replace the specified color plane.

## **void ColorImage::ReplaceHSISaturationPlane ( MonoImage \* plane )**

Replace the Saturation color plane in a HSI image with a **MonoImage**.

### Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a **MonoImage** that will replace the specified color plane.

## **void ColorImage::ReplaceHSLHuePlane ( MonoImage \* plane )**

Replace the Hue color plane in a HSL image with a **MonoImage**.

## Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a [MonoImage](#) that will replace the specified color plane.

### **void ColorImage::ReplaceHLSaturationPlane ( MonoImage \* plane )**

Replace the Saturation color plane in an HSL image with a [MonoImage](#).

## Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a [MonoImage](#) that will replace the specified color plane.

### **void ColorImage::ReplaceHSVHuePlane ( MonoImage \* plane )**

Replace the Hue color plane in a HSV image with a [MonoImage](#).

## Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a [MonoImage](#) that will replace the specified color plane.

### **void ColorImage::ReplaceHVSaturationPlane ( MonoImage \* plane )**

Replace the Saturation color plane in a HSV image with a [MonoImage](#).

## Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a [MonoImage](#) that will replace the specified color plane.

### **void ColorImage::ReplaceIntensityPlane ( MonoImage \* plane )**

Replace the Intensity color plane in a HSI image with a [MonoImage](#).

## Parameters:

**mode** The color mode in which to operate.

**plane** A pointer to a [MonoImage](#) that will replace the specified color plane.

## **void ColorImage::ReplaceLuminancePlane ( MonoImage \* plane )**

Replace the Luminance color plane in an HSL image with a **MonoImage**.

### **Parameters:**

**mode** The color mode in which to operate.

**plane** A pointer to a **MonoImage** that will replace the specified color plane.

## **void ColorImage::ReplaceRedPlane ( MonoImage \* plane )**

Replace the red color plane with a **MonoImage**.

### **Parameters:**

**mode** The color mode in which to operate.

**plane** A pointer to a **MonoImage** that will replace the specified color plane.

## **void ColorImage::ReplaceValuePlane ( MonoImage \* plane )**

Replace the Value color plane in an HSV with a **MonoImage**.

### **Parameters:**

**mode** The color mode in which to operate.

**plane** A pointer to a **MonoImage** that will replace the specified color plane.

## **BinaryImage \* ColorImage::ThresholdHSI ( Threshold & t )**

Perform a threshold in HSI space.

### **Parameters:**

**threshold** a reference to the **Threshold** object to use.

### **Returns:**

A pointer to a **BinaryImage** that represents the result of the threshold operation.

## **BinaryImage \* ColorImage::ThresholdHSI ( int hueLow, int hueHigh, int saturationLow,**

```
int saturationHigh,  
int intensityLow,  
int intensityHigh
```

```
)
```

Perform a threshold in HSI space.

#### Parameters:

<b>hueLow</b>	Low value for hue
<b>hueHigh</b>	High value for hue
<b>saturationLow</b>	Low value for saturation
<b>saturationHigh</b>	High value for saturation
<b>valueLow</b>	Low intensity
<b>valueHigh</b>	High intensity

#### Returns:

a pointer to a **BinaryImage** that represents the result of the threshold operation.

```
BinaryImage * ColorImage::ThresholdHSL ( int hueLow,  
                                         int hueHigh,  
                                         int saturationLow,  
                                         int saturationHigh,  
                                         int luminenceLow,  
                                         int luminenceHigh
```

```
)
```

Perform a threshold in HSL space.

#### Parameters:

<b>hueLow</b>	Low value for hue
<b>hueHigh</b>	High value for hue
<b>saturationLow</b>	Low value for saturation
<b>saturationHigh</b>	High value for saturation
<b>luminenceLow</b>	Low value for luminence
<b>luminenceHigh</b>	High value for luminence

#### Returns:

a pointer to a **BinaryImage** that represents the result of the threshold

operation.

## **BinaryImage \* ColorImage::ThresholdHSL ( Threshold & t )**

Perform a threshold in HSL space.

### **Parameters:**

**threshold** a reference to the **Threshold** object to use.

### **Returns:**

A pointer to a **BinaryImage** that represents the result of the threshold operation.

## **BinaryImage \* ColorImage::ThresholdHSV ( Threshold & t )**

Perform a threshold in HSV space.

### **Parameters:**

**threshold** a reference to the **Threshold** object to use.

### **Returns:**

A pointer to a **BinaryImage** that represents the result of the threshold operation.

## **BinaryImage \* ColorImage::ThresholdHSV ( int hueLow, int hueHigh, int saturationLow, int saturationHigh, int valueLow, int valueHigh )**

Perform a threshold in HSV space.

### **Parameters:**

**hueLow** Low value for hue

**hueHigh** High value for hue

**saturationLow** Low value for saturation

**saturationHigh** High value for saturation

**valueLow** Low value

**valueHigh** High value

### Returns:

a pointer to a **BinaryImage** that represents the result of the threshold operation.

## **BinaryImage \* ColorImage::ThresholdRGB ( Threshold & t )**

Perform a threshold in RGB space.

### Parameters:

**threshold** a reference to the **Threshold** object to use.

### Returns:

A pointer to a **BinaryImage** that represents the result of the threshold operation.

## **BinaryImage \* ColorImage::ThresholdRGB ( int redLow, int redHigh, int greenLow, int greenHigh, int blueLow, int blueHigh )**

Perform a threshold in RGB space.

### Parameters:

**redLow** Red low value

**redHigh** Red high value

**greenLow** Green low value

**greenHigh** Green high value

**blueLow** Blue low value

**blueHigh** Blue high value

### Returns:

A pointer to a **BinaryImage** that represents the result of the threshold

operation.

---

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

- C:/WindRiver/workspace/WPILib/Vision/**ColorImage.h**
- C:/WindRiver/workspace/WPILib/Vision/ColorImage.cpp

---

Generated by  1.7.2



# ColorReport\_struct Struct Reference

---

Tracking functions return this structure. [More...](#)

```
#include <VisionAPI.h>
```

[List of all members.](#)

## Public Attributes

```
int numberParticlesFound
int largestParticleNumber
float particleHueMax
float particleHueMin
float particleHueMean
float particleSatMax
float particleSatMin
float particleSatMean
float particleLumMax
float particleLumMin
float particleLumMean
```

---

## Detailed Description

Tracking functions return this structure.

---

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

- C:/WindRiver/workspace/WPILib/Vision2009/**VisionAPI.h**

[Class List](#)[Class Hierarchy](#)[Class Members](#)

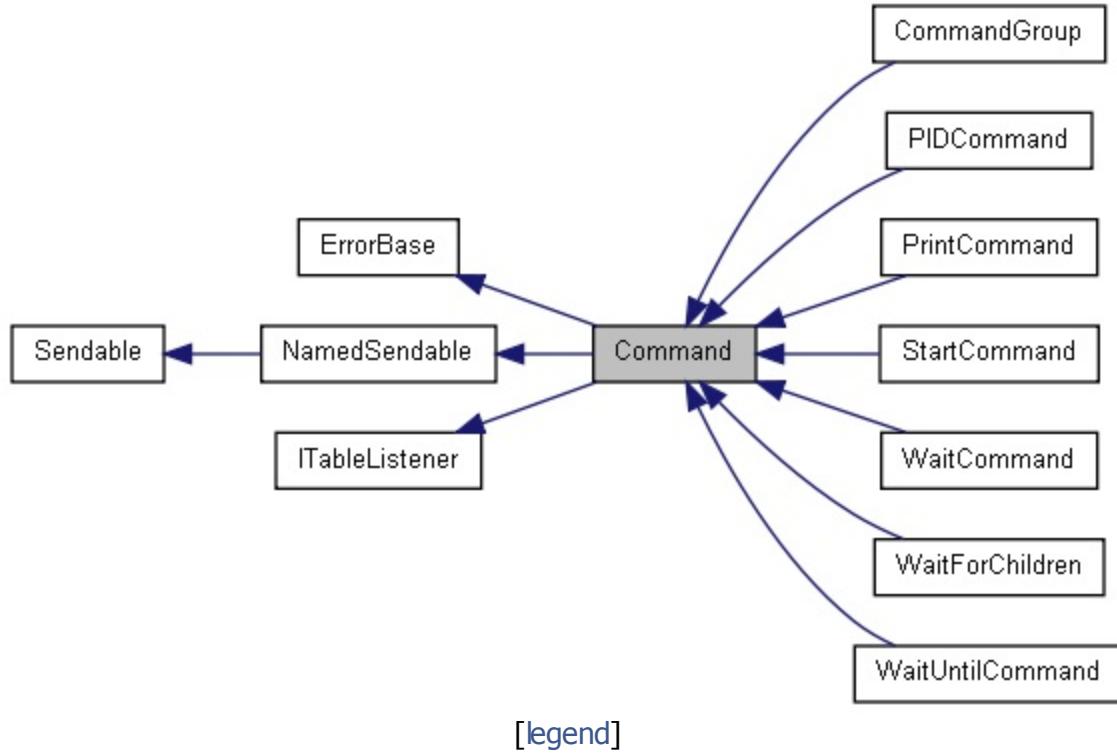
[Public Types](#) | [Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#) | [Friends](#)

# Command Class Reference

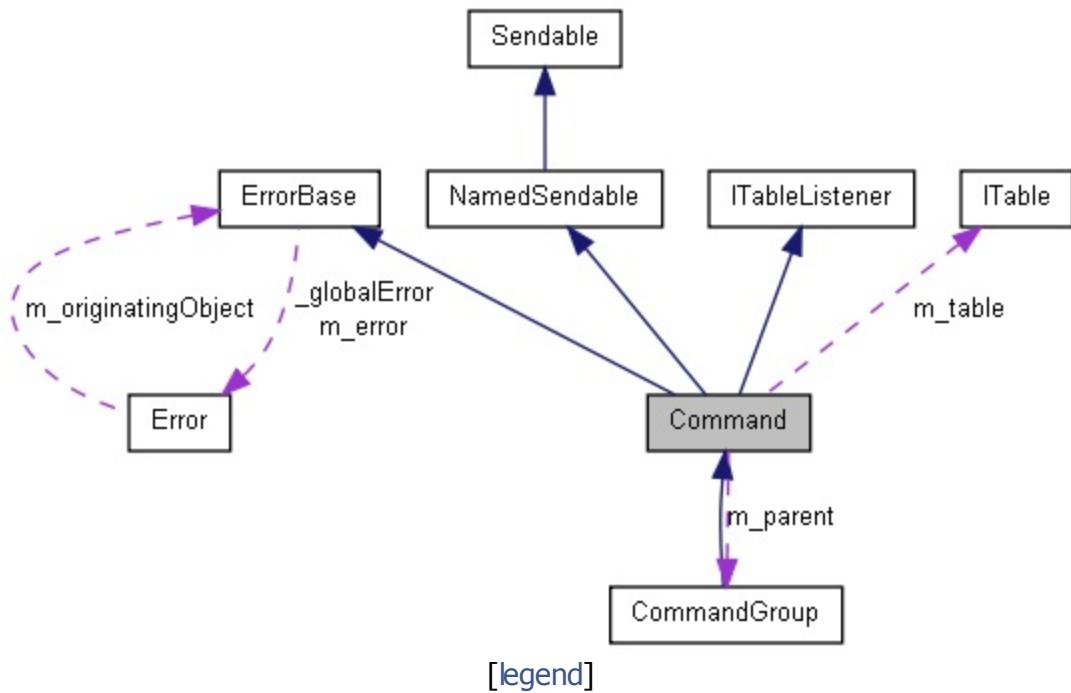
The **Command** class is at the very core of the entire command framework. [More...](#)

```
#include <Command.h>
```

Inheritance diagram for Command:



Collaboration diagram for Command:



List of all members.

## Public Types

```
typedef std::set< Subsystem * > SubsystemSet
```

# Public Member Functions

## **Command ()**

Creates a new command.

## **Command (const char \*name)**

Creates a new command with the given name and no timeout.

## **Command (double timeout)**

Creates a new command with the given timeout and a default name.

## **Command (const char \*name, double timeout)**

Creates a new command with the given name and timeout.

### **double TimeSinceInitialized ()**

Returns the time since this command was initialized (in seconds).

### **void Requires (Subsystem \*s)**

This method specifies that the given **Subsystem** is used by this command.

### **bool IsCanceled ()**

Returns whether or not this has been canceled.

### **void Start ()**

Starts up the command.

### **bool Run ()**

The run method is used internally to actually run the commands.

### **void Cancel ()**

This will cancel the current command.

### **bool IsRunning ()**

Returns whether or not the command is running.

### **bool IsInterruptible ()**

Returns whether or not this command can be interrupted.

### **void SetInterruptible (bool interruptible)**

Sets whether or not this command can be interrupted.

### **bool DoesRequire (Subsystem \*subsystem)**

Checks if the command requires the given

## Subsystem.

SubsystemSet	<b>GetRequirements ()</b> Returns the requirements (as an std::set of <b>Subsystems</b> pointers) of this command.
<b>CommandGroup</b> *	<b>GetGroup ()</b> Returns the <b>CommandGroup</b> that this command is a part of.
void	<b>SetRunWhenDisabled</b> (bool run) Sets whether or not this <b>Command</b> should run when the robot is disabled.
bool	<b>WillRunWhenDisabled ()</b> Returns whether or not this <b>Command</b> will run when the robot is disabled, or if it will cancel itself.
virtual std::string	<b>GetName ()</b>
virtual void	<b>InitTable</b> ( <b>ITable</b> *table) Initializes a table for this sendable object.
virtual <b>ITable</b> *	<b>GetTable ()</b>
virtual std::string	<b>GetSmartDashboardType ()</b>
virtual void	<b>ValueChanged</b> ( <b>ITable</b> *source, const std::string &key, <b>EntryValue</b> value, bool isNew) Called when a key-value pair is changed in a <b>ITable</b> <b>WARNING:</b> If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

# Protected Member Functions

void **SetTimeout** (double timeout)

Sets the timeout of this command.

bool **IsTimedOut** ()

Returns whether or not the **timeSinceInitialized()** method returns a number which is greater than or equal to the timeout for the command.

bool **AssertUnlocked** (const char \*message)

If changes are locked, then this will generate a CommandIllegalUse error.

void **SetParent** (**CommandGroup** \*parent)

Sets the parent of this command.

virtual void **Initialize** ()=0

The initialize method is called the first time this **Command** is run after being started.

virtual void **Execute** ()=0

The execute method is called repeatedly until this **Command** either finishes or is canceled.

virtual bool **IsFinished** ()=0

Returns whether this command is finished.

virtual void **End** ()=0

Called when the command ended peacefully.

virtual void **Interrupted** ()=0

Called when the command ends because somebody called **cancel()** or another command shared the same requirements as this one, and booted it out.

virtual void **\_Initialize** ()

virtual void **\_Interrupted** ()

virtual void **\_Execute** ()

virtual void **\_End** ()

virtual void **\_Cancel** ()

This works like **cancel()**, except that it doesn't throw an exception if it is a part of a command group.

# Protected Attributes

**ITable \* m\_table**

class **CommandGroup**  
class **Scheduler**

## Detailed Description

The **Command** class is at the very core of the entire command framework.

Every command can be started with a call to **Start()**. Once a command is started it will call **Initialize()**, and then will repeatedly call **Execute()** until the **IsFinished()** returns true. Once it does, **End()** will be called.

However, if at any point while it is running **Cancel()** is called, then the command will be stopped and **Interrupted()** will be called.

If a command uses a **Subsystem**, then it should specify that it does so by calling the **Requires(...)** method in its constructor. Note that a **Command** may have multiple requirements, and **Requires(...)** should be called for each one.

If a command is running and a new command with shared requirements is started, then one of two things will happen. If the active command is interruptible, then **Cancel()** will be called and the command will be removed to make way for the new one. If the active command is not interruptible, the other one will not even be started, and the active one will continue functioning.

### See also:

[CommandGroup](#)  
[Subsystem](#)

---

# Constructor & Destructor Documentation

## Command::Command ( )

Creates a new command.

The name of this command will be default.

## Command::Command ( const char \* name )

Creates a new command with the given name and no timeout.

### Parameters:

**name** the name for this command

## Command::Command ( double timeout )

Creates a new command with the given timeout and a default name.

### Parameters:

**timeout** the time (in seconds) before this command "times out"

### See also:

Command::isTimedOut() isTimedOut()

## Command::Command ( const char \* name,                           double          **timeout**                           )

Creates a new command with the given name and timeout.

### Parameters:

**name** the name of the command

**timeout** the time (in seconds) before this command "times out"

### See also:

Command::isTimedOut() isTimedOut()

# Member Function Documentation

## **void Command::\_Cancel( )** [protected, virtual]

This works like cancel(), except that it doesn't throw an exception if it is a part of a command group.

Should only be called by the parent command group.

## **bool Command::AssertUnlocked ( const char \* message )** [protected]

If changes are locked, then this will generate a CommandIllegalUse error.

### **Parameters:**

**message** the message to report on error (it is appended by a default message)

### **Returns:**

true if assert passed, false if assert failed

## **void Command::Cancel( )**

This will cancel the current command.

This will cancel the current command eventually. It can be called multiple times. And it can be called when the command is not running. If the command is running though, then the command will be marked as canceled and eventually removed.

A command can not be canceled if it is a part of a command group, you must cancel the command group instead.

## **bool Command::DoesRequire ( Subsystem \* system )**

Checks if the command requires the given **Subsystem**.

### **Parameters:**

**system** the system

### **Returns:**

whether or not the subsystem is required (false if given NULL)

## **virtual void Command::End ( )** [protected, pure virtual]

Called when the command ended peacefully.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Implemented in **CommandGroup**, **PrintCommand**, **StartCommand**, **WaitCommand**, **WaitForChildren**, and **WaitUntilCommand**.

## **CommandGroup \* Command::GetGroup ( )**

Returns the **CommandGroup** that this command is a part of.

Will return null if this **Command** is not in a group.

### **Returns:**

the **CommandGroup** that this command is a part of (or null if not in group)

## **std::string Command::GetName ( )** [virtual]

### **Returns:**

the name of the subtable of **SmartDashboard** that the **Sendable** object will use

Implements **NamedSendable**.

## **Command::SubsystemSet Command::GetRequirements ( )**

Returns the requirements (as an std::set of **Subsystems** pointers) of this command.

### **Returns:**

the requirements (as an std::set of **Subsystems** pointers) of this command

## **std::string Command::GetSmartDashboardType ( )** [virtual]

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

Reimplemented in **PIDCommand**.

### **ITable \* Command::GetTable( ) [virtual]**

#### **Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

### **void Command::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

#### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

Reimplemented in **PIDCommand**.

### **virtual void Command::Interrupted( ) [protected, pure virtual]**

Called when the command ends because somebody called **cancel()** or another command shared the same requirements as this one, and booted it out.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Generally, it is useful to simply call the **end()** method within this method

Implemented in **CommandGroup**, **PrintCommand**, **StartCommand**, **WaitCommand**, **WaitForChildren**, and **WaitUntilCommand**.

### **bool Command::IsCanceled( )**

Returns whether or not this has been canceled.

#### **Returns:**

whether or not this has been canceled

## **virtual bool Command::IsFinished( )** [protected, pure virtual]

Returns whether this command is finished.

If it is, then the command will be removed and [end\(\)](#) will be called.

It may be useful for a team to reference the [isTimedOut\(\)](#) method for time-sensitive commands.

### **Returns:**

whether this command is finished.

### **See also:**

[Command::isTimedOut\(\)](#) [isTimedOut\(\)](#)

Implemented in [CommandGroup](#), [PrintCommand](#), [StartCommand](#), [WaitCommand](#), [WaitForChildren](#), and [WaitUntilCommand](#).

## **bool Command::IsInterruptible( )**

Returns whether or not this command can be interrupted.

### **Returns:**

whether or not this command can be interrupted

Reimplemented in [CommandGroup](#).

## **bool Command::IsRunning( )**

Returns whether or not the command is running.

This may return true even if the command has just been canceled, as it may not have yet called [Command#interrupted\(\)](#).

### **Returns:**

whether or not the command is running

## **bool Command::IsTimedOut( )** [protected]

Returns whether or not the **timeSinceInitialized()** method returns a number which is greater than or equal to the timeout for the command.

If there is no timeout, this will always return false.

### Returns:

whether the time has expired

## **void Command::Requires ( Subsystem \* subsystem )**

This method specifies that the given **Subsystem** is used by this command.

This method is crucial to the functioning of the **Command** System in general.

Note that the recommended way to call this method is in the constructor.

### Parameters:

**subsystem** the **Subsystem** required

### See also:

**Subsystem**

## **bool Command::Run ( )**

The run method is used internally to actually run the commands.

### Returns:

whether or not the command should stay within the **Scheduler**.

## **void Command::SetInterruptible ( bool interruptible )**

Sets whether or not this command can be interrupted.

### Parameters:

**interruptible** whether or not this command can be interrupted

## **void Command::SetParent ( CommandGroup \* parent ) [protected]**

Sets the parent of this command.

No actual change is made to the group.

## Parameters:

**parent** the parent

### **void Command::SetRunWhenDisabled ( bool run )**

Sets whether or not this **Command** should run when the robot is disabled.

By default a command will not run when the robot is disabled, and will in fact be canceled.

## Parameters:

**run** whether or not this command should run when the robot is disabled

### **void Command::SetTimeout ( double timeout ) [protected]**

Sets the timeout of this command.

## Parameters:

**timeout** the timeout (in seconds)

## See also:

Command::isTimedOut() isTimedOut()

### **void Command::Start( )**

Starts up the command.

Gets the command ready to start.

Note that the command will eventually start, however it will not necessarily do so immediately, and may in fact be canceled before initialize is even called.

### **double Command::TimeSinceInitialized ( )**

Returns the time since this command was initialized (in seconds).

This function will work even if there is no specified timeout.

## Returns:

the time since this command was initialized (in seconds).

```
void Command::ValueChanged ( ITable * source,
                            const std::string & key,
                            EntryValue value,
                            bool isNew
                          ) [virtual]
```

Called when a key-value pair is changed in a **ITable** **WARNING:** If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

## Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

## bool Command::WillRunWhenDisabled ( )

Returns whether or not this **Command** will run when the robot is disabled, or if it will cancel itself.

## Returns:

whether or not this **Command** will run when the robot is disabled, or if it will cancel itself

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

- C:/WindRiver/workspace/WPIlib/Commands/**Command.h**
- C:/WindRiver/workspace/WPIlib/Commands/Command.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

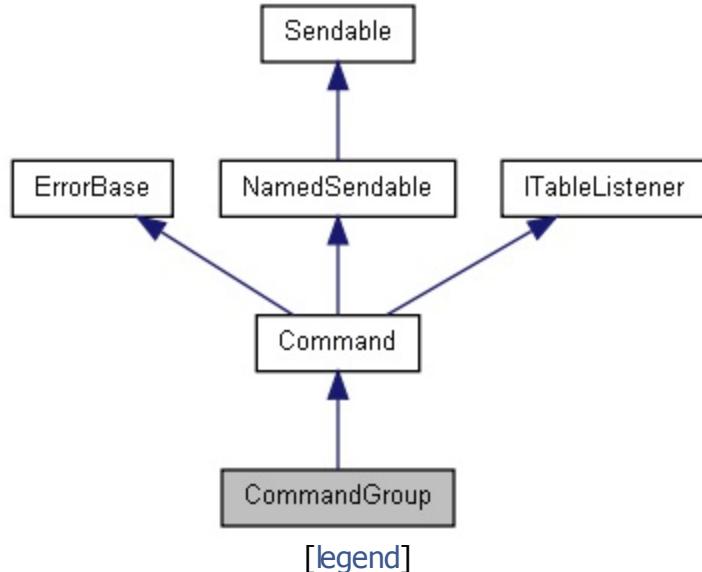
[Public Member Functions](#) |  
[Protected Member Functions](#)

# CommandGroup Class Reference

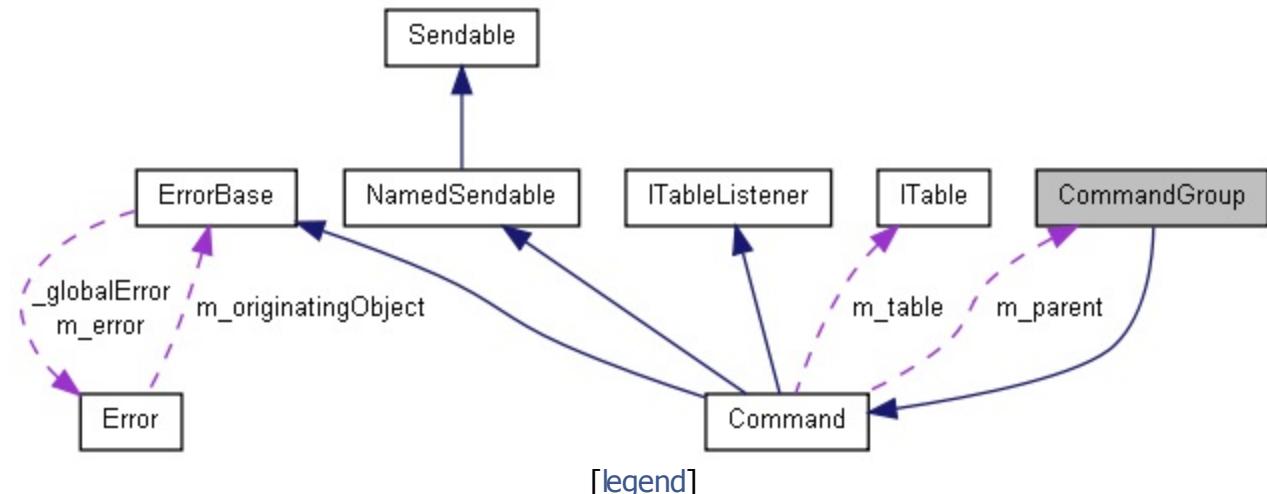
A **CommandGroup** is a list of commands which are executed in sequence. More...

```
#include <CommandGroup.h>
```

Inheritance diagram for CommandGroup:



Collaboration diagram for CommandGroup:



List of all members.

# Public Member Functions

## **CommandGroup ()**

Creates a new **CommandGroup**.

## **CommandGroup (const char \*name)**

Creates a new **CommandGroup** with the given name.

### **void AddSequential (Command \*command)**

Adds a new **Command** to the group.

### **void AddSequential (Command \*command, double timeout)**

Adds a new **Command** to the group with a given timeout.

### **void AddParallel (Command \*command)**

Adds a new child **Command** to the group.

### **void AddParallel (Command \*command, double timeout)**

Adds a new child **Command** to the group with the given timeout.

### **bool IsInterruptible ()**

Returns whether or not this command can be interrupted.

### **int GetSize ()**

# Protected Member Functions

virtual void **Initialize** ()

The initialize method is called the first time this **Command** is run after being started.

virtual void **Execute** ()

The execute method is called repeatedly until this **Command** either finishes or is canceled.

virtual bool **IsFinished** ()

Returns whether this command is finished.

virtual void **End** ()

Called when the command ended peacefully.

virtual void **Interrupted** ()

Called when the command ends because somebody called **cancel()** or another command shared the same requirements as this one, and booted it out.

virtual void **\_Initialize** ()

virtual void **\_Interrupted** ()

virtual void **\_Execute** ()

virtual void **\_End** ()

## Detailed Description

A **CommandGroup** is a list of commands which are executed in sequence.

Commands in a **CommandGroup** are added using the **AddSequential(...)** method and are called sequentially. **CommandGroups** are themselves **Commands** and can be given to other **CommandGroups**.

**CommandGroups** will carry all of the requirements of their **subcommands**.

Additional requirements can be specified by calling **Requires(...)** normally in the constructor.

CommandGroups can also execute commands in parallel, simply by adding them using **AddParallel(...)**.

**See also:**

**Command**  
**Subsystem**

---

# Constructor & Destructor Documentation

## **CommandGroup::CommandGroup ( const char \* name )**

Creates a new **CommandGroup** with the given name.

### **Parameters:**

**name** the name for this command group

# Member Function Documentation

```
void CommandGroup::AddParallel( Command * command,
                                double           timeout
                               )
```

Adds a new child **Command** to the group with the given timeout.

The **Command** will be started after all the previously added **Commands**.

Once the **Command** is started, it will run until it finishes, is interrupted, or the time expires, whichever is sooner. Note that the given **Command** will have no knowledge that it is on a timer.

Instead of waiting for the child to finish, a **CommandGroup** will have it run at the same time as the subsequent **Commands**. The child will run until either it finishes, the timeout expires, a new child with conflicting requirements is started, or the main sequence runs a **Command** with conflicting requirements. In the latter two cases, the child will be canceled even if it says it can't be interrupted.

Note that any requirements the given **Command** has will be added to the group. For this reason, a **Command's** requirements can not be changed after being added to a group.

It is recommended that this method be called in the constructor.

## Parameters:

**command** The command to be added  
**timeout** The timeout (in seconds)

```
void CommandGroup::AddParallel( Command * command )
```

Adds a new child **Command** to the group.

The **Command** will be started after all the previously added **Commands**.

Instead of waiting for the child to finish, a **CommandGroup** will have it run at the same time as the subsequent **Commands**. The child will run until either it finishes, a new child with conflicting requirements is started, or the main sequence runs a **Command** with conflicting requirements. In the latter two cases, the child will be canceled even if it says it can't be interrupted.

Note that any requirements the given **Command** has will be added to the group. For this reason, a **Command's** requirements can not be changed after being added to a group.

It is recommended that this method be called in the constructor.

#### Parameters:

**command** The command to be added

```
void CommandGroup::AddSequential( Command * command,  
                                 double      timeout  
                               )
```

Adds a new **Command** to the group with a given timeout.

The **Command** will be started after all the previously added commands.

Once the **Command** is started, it will be run until it finishes or the time expires, whichever is sooner. Note that the given **Command** will have no knowledge that it is on a timer.

Note that any requirements the given **Command** has will be added to the group. For this reason, a **Command's** requirements can not be changed after being added to a group.

It is recommended that this method be called in the constructor.

#### Parameters:

**command** The **Command** to be added

**timeout** The timeout (in seconds)

```
void CommandGroup::AddSequential( Command * command )
```

Adds a new **Command** to the group.

The **Command** will be started after all the previously added **Commands**.

Note that any requirements the given **Command** has will be added to the group. For this reason, a **Command's** requirements can not be changed after being added to a group.

It is recommended that this method be called in the constructor.

## Parameters:

**command** The [Command](#) to be added

### **void CommandGroup::End( )** [protected, virtual]

Called when the command ended peacefully.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Implements [Command](#).

### **void CommandGroup::Interrupted( )** [protected, virtual]

Called when the command ends because somebody called [cancel\(\)](#) or another command shared the same requirements as this one, and booted it out.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Generally, it is useful to simply call the [end\(\)](#) method within this method

Implements [Command](#).

### **bool CommandGroup::IsFinished( )** [protected, virtual]

Returns whether this command is finished.

If it is, then the command will be removed and [end\(\)](#) will be called.

It may be useful for a team to reference the [isTimedOut\(\)](#) method for time-sensitive commands.

#### >Returns:

whether this command is finished.

#### See also:

[Command::isTimedOut\(\)](#) [isTimedOut\(\)](#)

Implements [Command](#).

## **bool CommandGroup::IsInterruptible( )**

Returns whether or not this command can be interrupted.

### **Returns:**

whether or not this command can be interrupted

Reimplemented from **Command**.

---

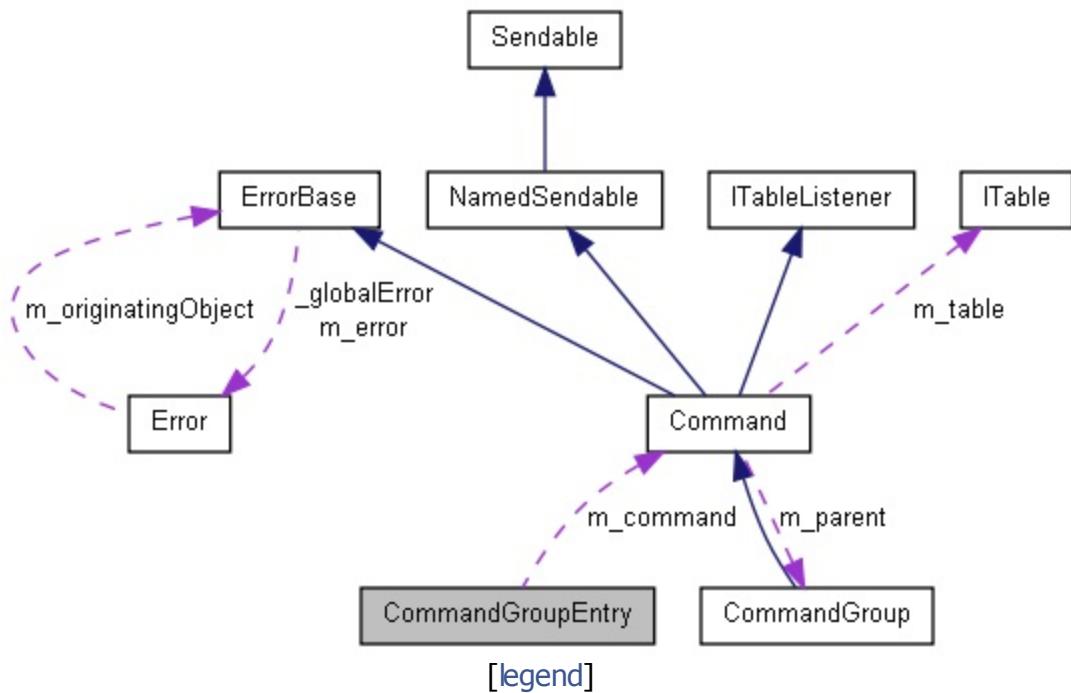
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/Commands/**CommandGroup.h**
- C:/WindRiver/workspace/WPILib/Commands/CommandGroup.cpp



# CommandGroupEntry Class Reference

Collaboration diagram for CommandGroupEntry:



List of all members.

## Public Types

enum **Sequence** { **kSequence\_InSequence**, **kSequence\_BranchPeer**,  
**kSequence\_BranchChild** }

**CommandGroupEntry** (**Command** \*command, Sequence state)

**CommandGroupEntry** (**Command** \*command, Sequence state,  
double timeout)

bool **IsTimedOut** ()

double **m\_timeout**

**Command** \* **m\_command**

## Sequence **m\_state**

---

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

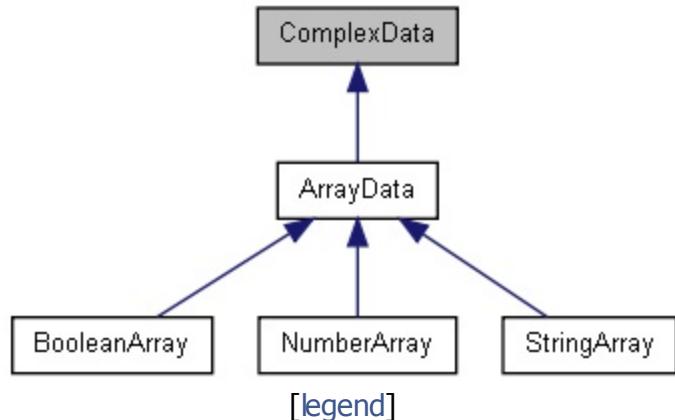
- C:/WindRiver/workspace/WPILib/Commands/**CommandGroupEntry.h**
- C:/WindRiver/workspace/WPILib/Commands/CommandGroupEntry.cpp

Generated by  1.7.2

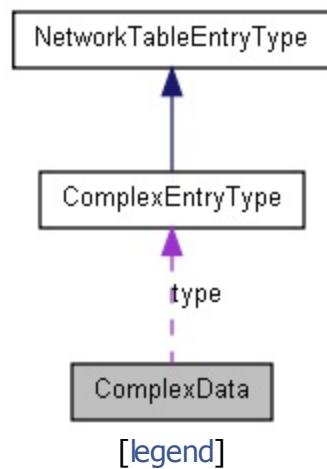


# ComplexData Class Reference

Inheritance diagram for ComplexData:



Collaboration diagram for ComplexData:



List of all members.

# Public Member Functions

**ComplexData ([ComplexEntryType](#) &type)**

**ComplexEntryType & [GetType](#) ()**

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

- C:/WindRiver/workspace/WPILib/networktables2/type/[ComplexData.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/type/ComplexData.cpp

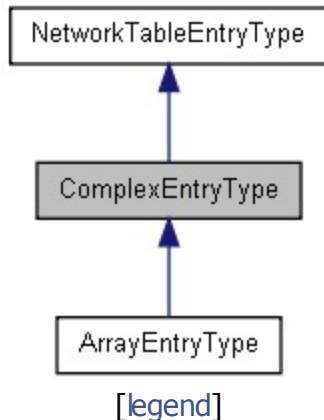
Generated by [doxygen](#) 1.7.2

[Class List](#)[Class Hierarchy](#)[Class Members](#)

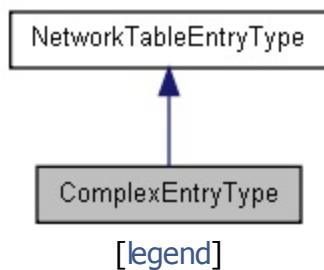
[Public Member Functions](#) |  
[Protected Member Functions](#)

# ComplexEntryType Class Reference

Inheritance diagram for ComplexEntryType:



Collaboration diagram for ComplexEntryType:



List of all members.

# Public Member Functions

virtual bool	<b>isComplex ()</b>
virtual <b>EntryValue</b>	<b>internalizeValue</b> (std::string &key, <b>ComplexData</b> &externalRepresentation, <b>EntryValue</b> currentInternalValue)=0
virtual void	<b>exportValue</b> (std::string &key, <b>EntryValue</b> internalData, <b>ComplexData</b> &externalRepresentation)=0

## **ComplexEntryType** (TypeId id, const char \*name)

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

- C:/WindRiver/workspace/WPILib/networktables2/type/**ComplexEntryType.h**
- C:/WindRiver/workspace/WPILib/networktables2/type/ComplexEntryType.cpp

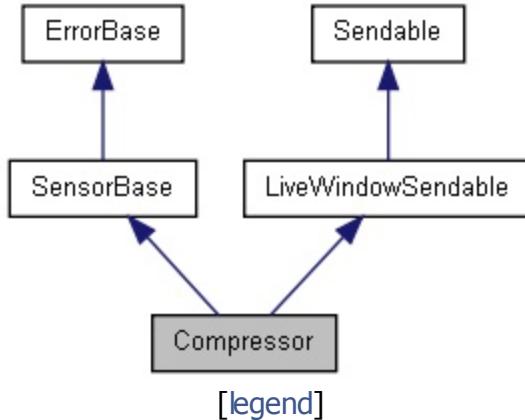


# Compressor Class Reference

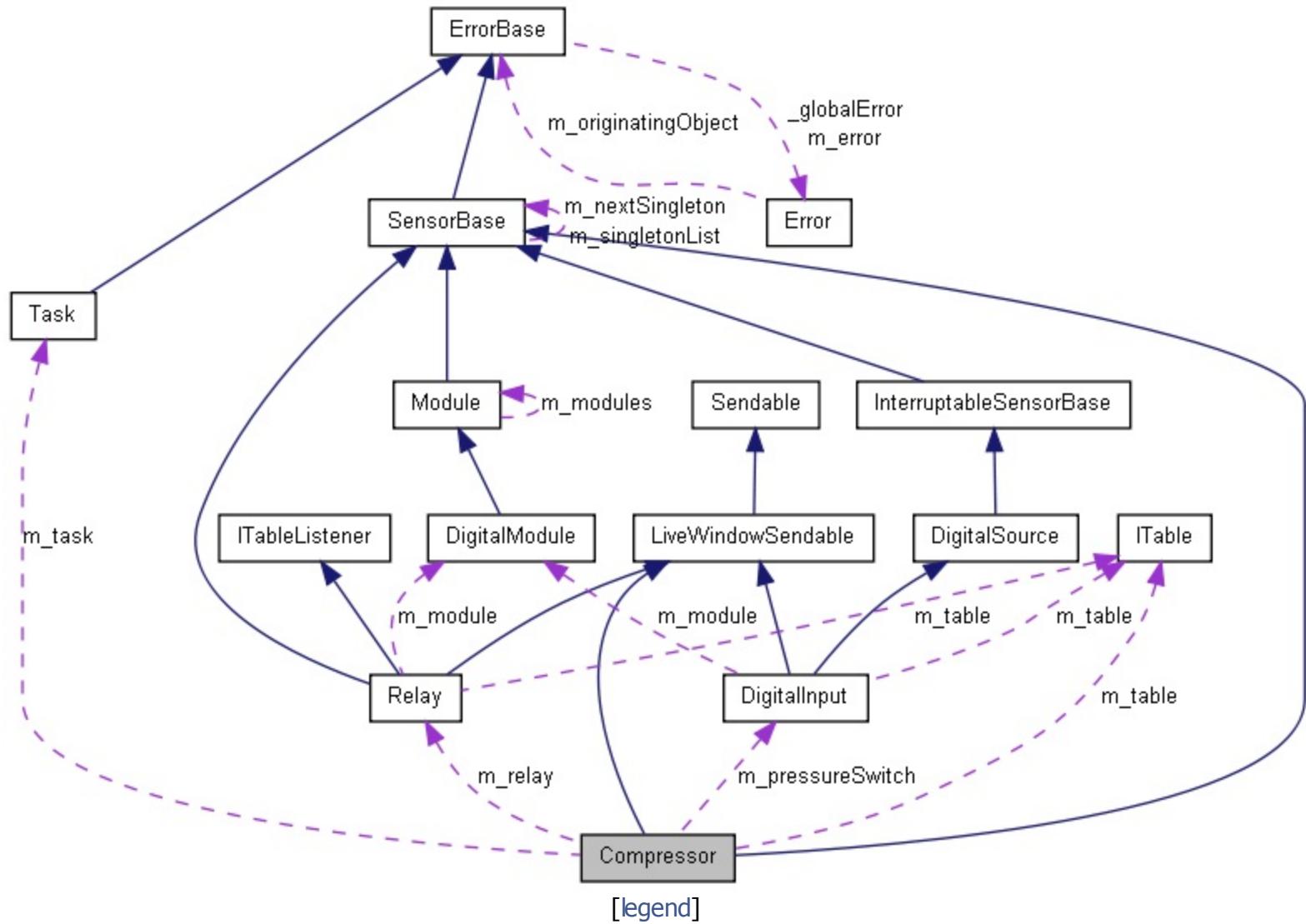
## **Compressor** object. More...

```
#include <Compressor.h>
```

## Inheritance diagram for Compressor:



## Collaboration diagram for Compressor:



## List of all members.

**Compressor** (UINT32 pressureSwitchChannel, UINT32 compressorRelayChannel)  
**Compressor** constructor.

**Compressor** (UINT8 pressureSwitchModuleNumber, UINT32 pressureSwitchChannel, UINT8 compressorRelayModuleNumber, UINT32 compressorRelayChannel)  
**Compressor** constructor.

**~Compressor ()**

Delete the **Compressor** object.

void **Start ()**

Start the compressor.

void **Stop ()**

Stop the compressor.

bool **Enabled ()**

Get the state of the enabled flag.

UINT32 **GetPressureSwitchValue ()**

Get the pressure switch value.

void **SetRelayValue (Relay::Value relayValue)**

Operate the relay for the compressor.

void **UpdateTable ()**

Update the table for this sendable object with the latest values.

void **StartLiveWindowMode ()**

Start having this sendable object automatically respond to value changes reflect the value on the table.

void **StopLiveWindowMode ()**

Stop having this sendable object automatically respond to value changes.

std::string **GetSmartDashboardType ()**

void **InitTable (ITable \*subTable)**

Initializes a table for this sendable object.

**ITable \* GetTable ()**

## Detailed Description

### Compressor object.

The **Compressor** object is designed to handle the operation of the compressor, pressure sensor and relay for a FIRST robot pneumatics system. The **Compressor** object starts a task which runs in the background and periodically polls the pressure sensor and operates the relay that controls the compressor.

---

# Constructor & Destructor Documentation

```
Compressor::Compressor ( UINT32 pressureSwitchChannel,  
                         UINT32 compressorRelayChannel  
                     )
```

**Compressor** constructor.

Given a relay channel and pressure switch channel (both in the default digital module), initialize the **Compressor** object.

You MUST start the compressor by calling the **Start()** method.

## Parameters:

<b>pressureSwitchChannel</b>	The GPIO channel that the pressure switch is attached to.
<b>compressorRelayChannel</b>	The relay channel that the compressor relay is attached to.

```
Compressor::Compressor ( UINT8 pressureSwitchModuleNumber,  
                         UINT32 pressureSwitchChannel,  
                         UINT8 compressorRelayModuleNumber,  
                         UINT32 compressorRelayChannel  
                     )
```

**Compressor** constructor.

Given a fully specified relay channel and pressure switch channel, initialize the **Compressor** object.

You MUST start the compressor by calling the **Start()** method.

## Parameters:

<b>pressureSwitchModuleNumber</b>	The digital module that the pressure switch is attached to.
<b>pressureSwitchChannel</b>	The GPIO channel that the pressure switch is attached to.
<b>compressorRelayModuleNumber</b>	The digital module that the compressor relay is attached to.
<b>compressorRelayChannel</b>	The relay channel that the compressor

relay is attached to.

## **Compressor::~Compressor( )**

Delete the **Compressor** object.

Delete the allocated resources for the compressor and kill the compressor task that is polling the pressure switch.

---

# Member Function Documentation

## **bool Compressor::Enabled( )**

Get the state of the enabled flag.

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

### **Returns:**

The state of the compressor thread's enable flag.

## **UINT32 Compressor::GetPressureSwitchValue( )**

Get the pressure switch value.

Read the pressure switch digital input.

### **Returns:**

The current state of the pressure switch.

## **std::string Compressor::GetSmartDashboardType( ) [virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

## **ITable \* Compressor::GetTable( ) [virtual]**

### **Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

## **void Compressor::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

## Parameters:

**subtable** The table to put the values in.

Implements **Sendable**.

### **void Compressor::SetRelayValue ( Relay::Value relayValue )**

Operate the relay for the compressor.

Change the value of the relay output that is connected to the compressor motor. This is only intended to be called by the internal polling thread.

### **void Compressor::Start ( )**

Start the compressor.

This method will allow the polling loop to actually operate the compressor. The is stopped by default and won't operate until starting it.

### **void Compressor::Stop ( )**

Stop the compressor.

This method will stop the compressor from turning on.

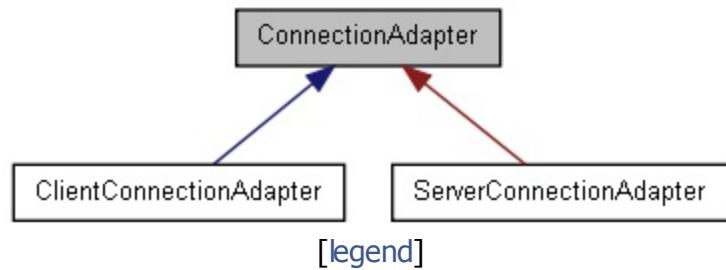
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/**Compressor.h**
- C:/WindRiver/workspace/WPILib/Compressor.cpp



# ConnectionAdapter Class Reference

Inheritance diagram for ConnectionAdapter:



List of all members.

## Public Member Functions

virtual void	<b>keepAlive ()=0</b>
virtual void	<b>clientHello (ProtocolVersion protocolRevision)=0</b>
virtual void	<b>serverHelloComplete ()=0</b>
virtual void	<b>protocolVersionUnsupported (ProtocolVersion protocolRevision)=0</b>
virtual void	<b>offerIncomingAssignment (NetworkTableEntry *newEntry)=0</b>
virtual void	<b>offerIncomingUpdate (NetworkTableEntry *newEntry, SequenceNumber sequenceNumber, EntryValue value)=0</b>
virtual NetworkTableEntry *	<b>GetEntry (EntryId)=0</b>
virtual void	<b>badMessage (BadMessageException &amp;e)=0</b> called if a bad message exception is thrown
virtual void	<b>ioException (IOException &amp;e)=0</b> called if an io exception is thrown

# Member Function Documentation

**virtual void ConnectionAdapter::badMessage ( BadMessageException & e )** [pure virtual]

called if a bad message exception is thrown

## Parameters:

e

Implemented in **ClientConnectionAdapter**, and **ServerConnectionAdapter**.

**virtual void ConnectionAdapter::ioException ( IOException & e )** [pure virtual]

called if an io exception is thrown

## Parameters:

e

Implemented in **ClientConnectionAdapter**, and **ServerConnectionAdapter**.

---

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

- C:/WindRiver/workspace/WPILib/networktables2/connection/**ConnectionAdapter.h**

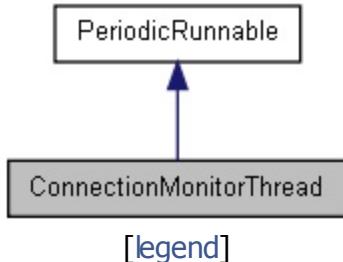


# ConnectionMonitorThread Class Reference

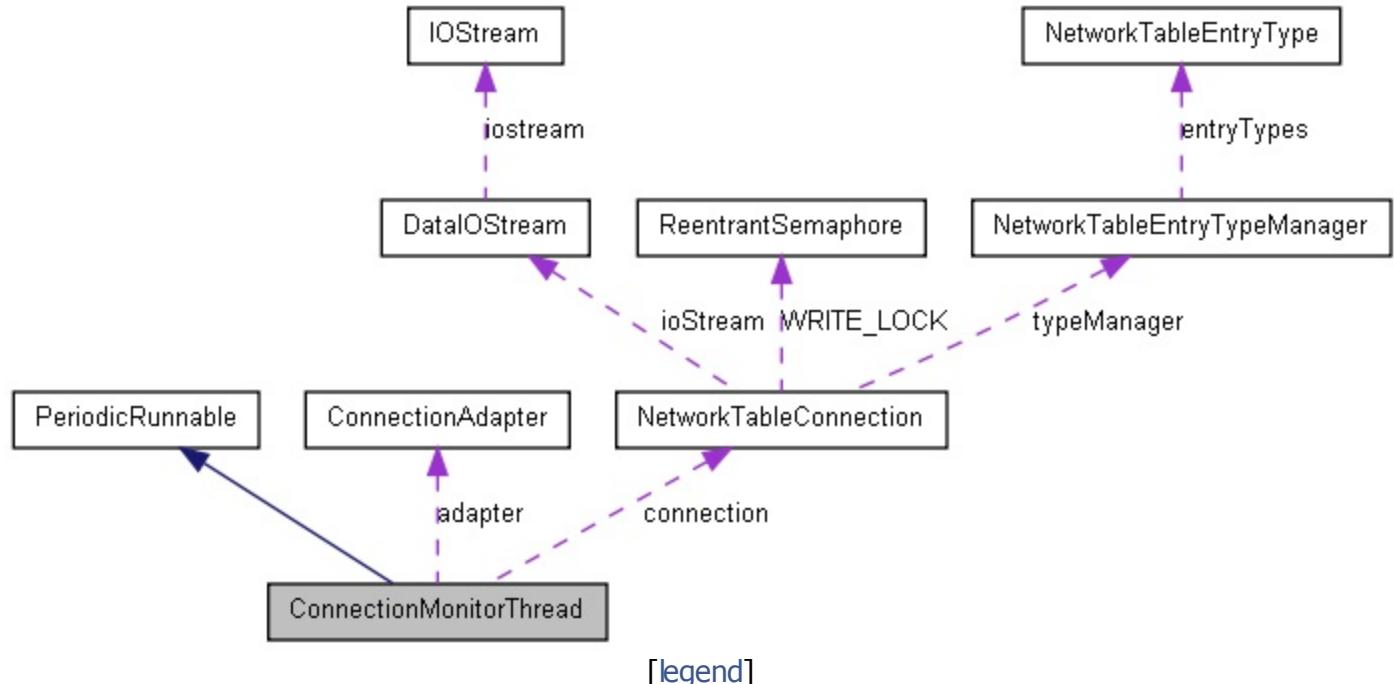
A periodic thread that repeatedly reads from a connection. [More...](#)

```
#include <ConnectionMonitorThread.h>
```

Inheritance diagram for ConnectionMonitorThread:



Collaboration diagram for ConnectionMonitorThread:



[List of all members.](#)

## Public Member Functions

**ConnectionMonitorThread** (**ConnectionAdapter** &adapter,  
**NetworkTableConnection** &connection)  
create a new monitor thread

void **run ()**

the method that will be called periodically on a thread

## Detailed Description

A periodic thread that repeatedly reads from a connection.

### Author:

Mitchell

---

# Constructor & Destructor Documentation

**ConnectionMonitorThread::ConnectionMonitorThread ( [ConnectionAdapter](#) &  
[NetworkTableConnecti](#)  
)**

create a new monitor thread

**Parameters:**

**adapter**

**connection**

# Member Function Documentation

## **void ConnectionMonitorThread::run( ) [virtual]**

the method that will be called periodically on a thread

### **Exceptions:**

thrown when the thread is supposed to be interrupted and **InterruptedException** stop (implementers should always let this exception fall through)

Implements **PeriodicRunnable**.

---

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

- C:/WindRiver/workspace/WPILib/networktables2/connection/**ConnectionMonitorT**
- C:/WindRiver/workspace/WPILib/networktables2/connection/ConnectionMonitorThre

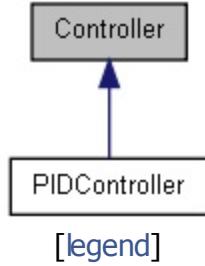


# Controller Class Reference

Interface for Controllers Common interface for controllers. [More...](#)

```
#include <Controller.h>
```

Inheritance diagram for Controller:



[List of all members.](#)

## Public Member Functions

virtual void **Enable ()=0**

Allows the control loop to run.

virtual void **Disable ()=0**

Stops the control loop from running until explicitly re-enabled by calling enable()

---

## Detailed Description

Interface for Controllers Common interface for controllers.

Controllers run control loops, the most common are PID controllers and their variants, but this includes anything that is controlling an actuator in a separate thread.

---

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

- C:/WindRiver/workspace/WPILib/**Controller.h**

Generated by  1.7.2

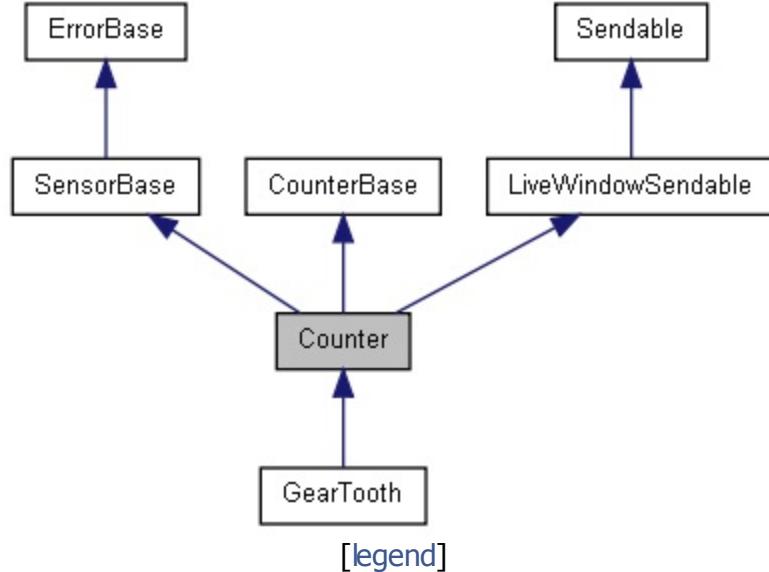


# Counter Class Reference

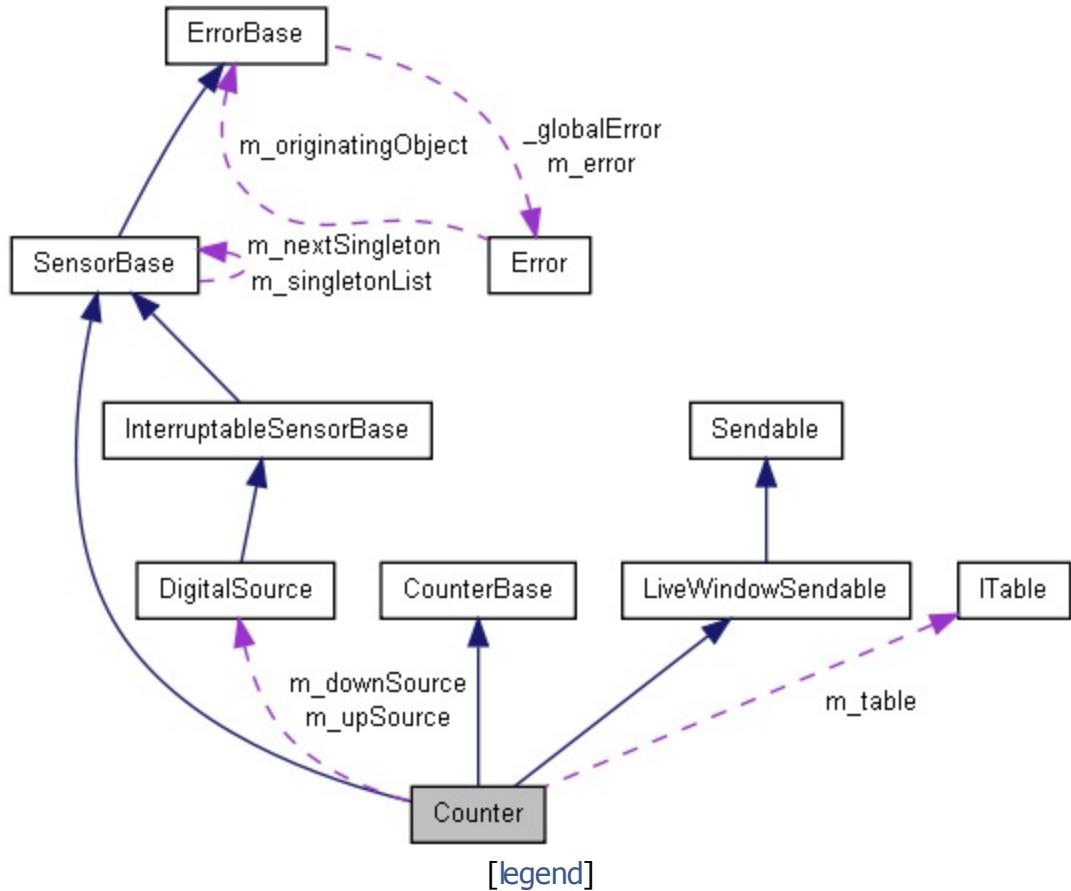
Class for counting the number of ticks on a digital input channel. [More...](#)

```
#include <Counter.h>
```

Inheritance diagram for Counter:



Collaboration diagram for Counter:



List of all members.

## Public Types

enum **Mode** { **kTwoPulse** = 0, **kSemiperiod** = 1, **kPulseLength** = 2,  
**kExternalDirection** = 3 }

# Public Member Functions

## **Counter ()**

Create an instance of a counter where no sources are selected.

## **Counter (UINT32 channel)**

Create an instance of a **Counter** object.

## **Counter (UINT8 moduleNumber, UINT32 channel)**

Create an instance of a **Counter** object.

## **Counter (DigitalSource \*source)**

Create an instance of a counter from a Digital Input.

## **Counter (DigitalSource &source)**

## **Counter (AnalogTrigger \*trigger)**

Create an instance of a **Counter** object.

## **Counter (AnalogTrigger &trigger)**

## **Counter (EncodingType encodingType, DigitalSource \*upSource,**

## **DigitalSource \*downSource, bool inverted)**

virtual **~Counter ()**

Delete the **Counter** object.

void **setUpSource (UINT32 channel)**

Set the upsouce for the counter as a digital input channel.

void **setUpSource (UINT8 moduleNumber, UINT32 channel)**

Set the up source for the counter as digital input channel and slot.

void **setUpSource (AnalogTrigger \*analogTrigger,**

AnalogTriggerOutput::Type triggerType)

Set the up counting source to be an analog trigger.

void **setUpSource (AnalogTrigger &analogTrigger,**

AnalogTriggerOutput::Type triggerType)

Set the up counting source to be an analog trigger.

void **setUpSource (DigitalSource \*source)**

Set the source object that causes the counter to count up.

void **setUpSource (DigitalSource &source)**

Set the source object that causes the counter to count up.

void **setUpSourceEdge (bool risingEdge, bool fallingEdge)**

Set the edge sensitivity on an up counting source.

void **clearUpSource ()**

Disable the up counting source to the counter.

void **setUpDownSource (UINT32 channel)**

void	<b>SetDownSource</b> (UINT8 moduleNumber, UINT32 channel)	Set the down counting source to be a digital input channel. Set the down counting source to be a digital input slot and channel.
void	<b>SetDownSource</b> (AnalogTrigger *analogTrigger, AnalogTriggerOutput::Type triggerType)	Set the down counting source to be an analog trigger.
void	<b>SetDownSource</b> (AnalogTrigger &analogTrigger, AnalogTriggerOutput::Type triggerType)	Set the down counting source to be an analog trigger.
void	<b>SetDownSource</b> (DigitalSource *source)	Set the source object that causes the counter to count down.
void	<b>SetDownSource</b> (DigitalSource &source)	Set the source object that causes the counter to count down.
void	<b>SetDownSourceEdge</b> (bool risingEdge, bool fallingEdge)	Set the edge sensitivity on a down counting source.
void	<b>ClearDownSource</b> ()	Disable the down counting source to the counter.
void	<b>setUpDownCounterMode</b> ()	Set standard up / down counting mode on this counter.
void	<b>SetExternalDirectionMode</b> ()	Set external direction mode on this counter.
void	<b>SetSemiPeriodMode</b> (bool highSemiPeriod)	Set Semi-period mode on this counter.
void	<b>SetPulseLengthMode</b> (float threshold)	Configure the counter to count in up or down based on the length of the input pulse.
void	<b>SetReverseDirection</b> (bool reverseDirection)	Set the <b>Counter</b> to return reversed sensing on the direction.
void	<b>Start</b> ()	Start the <b>Counter</b> counting.
INT32	<b>Get</b> ()	Read the current counter value.
void	<b>Reset</b> ()	Reset the <b>Counter</b> to zero.
void	<b>Stop</b> ()	Stop the <b>Counter</b> .
double	<b>GetPeriod</b> ()	

void	<b>SetMaxPeriod</b> (double maxPeriod)	Set the maximum period where the device is still considered "moving".
void	<b>SetUpdateWhenEmpty</b> (bool enabled)	Select whether you want to continue updating the event timer output when there are no samples captured.
bool	<b>GetStopped</b> ()	Determine if the clock is stopped.
bool	<b>GetDirection</b> ()	The last direction the counter value changed.
UINT32	<b>GetIndex</b> ()	
void	<b>UpdateTable</b> ()	Update the table for this sendable object with the latest values.
void	<b>StartLiveWindowMode</b> ()	Start having this sendable object automatically respond to value changes reflect the value on the table.
void	<b>StopLiveWindowMode</b> ()	Stop having this sendable object automatically respond to value changes.
virtual std::string	<b>GetSmartDashboardType</b> ()	
void	<b>InitTable</b> (ITable *subTable)	Initializes a table for this sendable object.
<b>ITable</b> *	<b>GetTable</b> ()	

## Detailed Description

Class for counting the number of ticks on a digital input channel.

This is a general purpose class for counting repetitive events. It can return the number of counts, the period of the most recent cycle, and detect when the signal being counted has stopped by supplying a maximum cycle time.

---

# Constructor & Destructor Documentation

## Counter::Counter( )

Create an instance of a counter where no sources are selected.

Then they all must be selected by calling functions to specify the upsource and the downsource independently.

## Counter::Counter( **UINT32 channel** ) [explicit]

Create an instance of a **Counter** object.

Create an up-Counter instance given a channel. The default digital module is assumed.

## Counter::Counter( **UINT8 moduleNumber,** **UINT32 channel** )

Create an instance of a **Counter** object.

Create an instance of an up-Counter given a digital module and a channel.

### Parameters:

- |                     |                                   |
|---------------------|-----------------------------------|
| <b>moduleNumber</b> | The digital module (1 or 2).      |
| <b>channel</b>      | The channel in the digital module |

## Counter::Counter( **DigitalSource \* source** ) [explicit]

Create an instance of a counter from a Digital Input.

This is used if an existing digital input is to be shared by multiple other objects such as encoders.

## Counter::Counter( **AnalogTrigger \* trigger** ) [explicit]

Create an instance of a **Counter** object.

Create an instance of a simple up-Counter given an analog trigger. Use the trigger state output from the analog trigger.

---

# Member Function Documentation

## **INT32 Counter::Get( ) [virtual]**

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.

Implements **CounterBase**.

## **bool Counter::GetDirection( ) [virtual]**

The last direction the counter value changed.

### **Returns:**

The last direction the counter value changed.

Implements **CounterBase**.

## **std::string Counter::GetSmartDashboardType( ) [virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

Reimplemented in **GearTooth**.

## **bool Counter::GetStopped( ) [virtual]**

Determine if the clock is stopped.

Determine if the clocked input is stopped based on the MaxPeriod value set using the SetMaxPeriod method. If the clock exceeds the MaxPeriod, then the device (and counter) are assumed to be stopped and it returns true.

### **Returns:**

Returns true if the most recent counter period exceeds the MaxPeriod value set by SetMaxPeriod.

Implements **CounterBase**.

### **ITable \* Counter::GetTable( ) [virtual]**

#### **Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

### **void Counter::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

#### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

### **void Counter::Reset( ) [virtual]**

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.

Implements **CounterBase**.

### **void Counter::SetDownSource( UINT32 channel )**

Set the down counting source to be a digital input channel.

The slot will be set to the default digital module slot.

### **void Counter::SetDownSource( UINT8 moduleNumber,                               UINT32 channel                               )**

Set the down counting source to be a digital input slot and channel.

## Parameters:

**moduleNumber** The digital module (1 or 2).

**channel** The digital channel (1..14).

```
void Counter::SetDownSource( AnalogTrigger * analogTrigger,  
                            AnalogTriggerOutput::Type triggerType  
                          )
```

Set the down counting source to be an analog trigger.

## Parameters:

**analogTrigger** The analog trigger object that is used for the Down Source

**triggerType** The analog trigger output that will trigger the counter.

```
void Counter::SetDownSource( DigitalSource * source )
```

Set the source object that causes the counter to count down.

Set the down counting **DigitalSource**.

```
void Counter::SetDownSource( AnalogTrigger & analogTrigger,  
                            AnalogTriggerOutput::Type triggerType  
                          )
```

Set the down counting source to be an analog trigger.

## Parameters:

**analogTrigger** The analog trigger object that is used for the Down Source

**triggerType** The analog trigger output that will trigger the counter.

```
void Counter::SetDownSource( DigitalSource & source )
```

Set the source object that causes the counter to count down.

Set the down counting **DigitalSource**.

```
void Counter::SetDownSourceEdge( bool risingEdge,
```

**bool fallingEdge**

)

Set the edge sensitivity on a down counting source.

Set the down source to either detect rising edges or falling edges.

### **void Counter::SetExternalDirectionMode( )**

Set external direction mode on this counter.

Counts are sourced on the Up counter input. The Down counter input represents the direction to count.

### **void Counter::SetMaxPeriod( double maxPeriod ) [virtual]**

Set the maximum period where the device is still considered "moving".

Sets the maximum period where the device is considered moving. This value is used to determine the "stopped" state of the counter using the GetStopped method.

#### **Parameters:**

**maxPeriod** The maximum period where the counted device is considered moving in seconds.

Implements **CounterBase**.

### **void Counter::SetPulseLengthMode( float threshold )**

Configure the counter to count in up or down based on the length of the input pulse.

This mode is most useful for direction sensitive gear tooth sensors.

#### **Parameters:**

**threshold** The pulse length beyond which the counter counts the opposite direction. Units are seconds.

### **void Counter::SetReverseDirection( bool reverseDirection )**

Set the **Counter** to return reversed sensing on the direction.

This allows counters to change the direction they are counting in the case of 1X and 2X quadrature encoding only. Any other counter mode isn't supported.

## Parameters:

**reverseDirection** true if the value counted should be negated.

```
void Counter::SetSemiPeriodMode ( bool highSemiPeriod )
```

Set Semi-period mode on this counter.

Counts up on both rising and falling edges.

**void Counter::SetUpdateWhenEmpty ( bool enabled )**

Select whether you want to continue updating the event timer output when there are no samples captured.

The output of the event timer has a buffer of periods that are averaged and posted to a register on the FPGA. When the timer detects that the event source has stopped (based on the MaxPeriod) the buffer of samples to be averaged is emptied. If you enable the update when empty, you will be notified of the stopped source and the event time will report 0 samples. If you disable update when empty, the most recent average will remain on the output until a new sample is acquired. You will never see 0 samples output (except when there have been no events since an FPGA reset) and you will likely not see the stopped bit become true (since it is updated at the end of an average and there are no samples to average).

**void Counter::SetUpDownCounterMode( )**

Set standard up / down counting mode on this counter.

Up and down counts are sourced independently from two inputs.

```
void Counter::SetUpSource( AnalogTrigger & analogTrigger,  
                           AnalogTriggerOutput::Type triggerType  
                         )
```

Set the up counting source to be an analog trigger.

### Parameters:

- analogTrigger** The analog trigger object that is used for the Up Source  
**triggerType** The analog trigger output that will trigger the counter.

```
void Counter::SetUpSource( UINT8 moduleNumber,
                           UINT32 channel
                         )
```

Set the up source for the counter as digital input channel and slot.

### Parameters:

- moduleNumber** The digital module (1 or 2).  
**channel** The digital channel (1..14).

```
void Counter::SetUpSource( DigitalSource * source )
```

Set the source object that causes the counter to count up.

Set the up counting **DigitalSource**.

```
void Counter::SetUpSource( AnalogTrigger * analogTrigger,
                           AnalogTriggerOutput::Type triggerType
                         )
```

Set the up counting source to be an analog trigger.

### Parameters:

- analogTrigger** The analog trigger object that is used for the Up Source  
**triggerType** The analog trigger output that will trigger the counter.

```
void Counter::SetUpSource( UINT32 channel )
```

Set the upsorce for the counter as a digital input channel.

The slot will be the default digital module slot.

## **void Counter::SetUpSource ( DigitalSource & source )**

Set the source object that causes the counter to count up.

Set the up counting **DigitalSource**.

## **void Counter::SetUpSourceEdge ( bool risingEdge, bool fallingEdge )**

Set the edge sensitivity on an up counting source.

Set the up source to either detect rising edges or falling edges.

## **void Counter::Start ( ) [virtual]**

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.

Implements **CounterBase**.

## **void Counter::Stop ( ) [virtual]**

Stop the **Counter**.

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

Implements **CounterBase**.

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

- C:/WindRiver/workspace/WPILib/**Counter.h**
- C:/WindRiver/workspace/WPILib/Counter.cpp

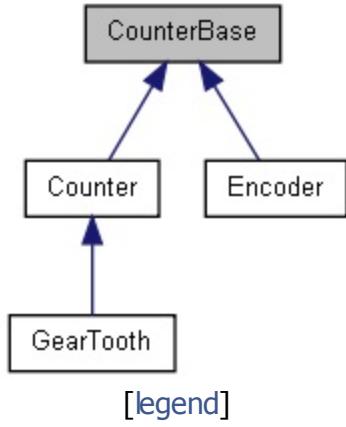


# CounterBase Class Reference

Interface for counting the number of ticks on a digital input channel. [More...](#)

```
#include <CounterBase.h>
```

Inheritance diagram for CounterBase:



[\[legend\]](#)

[List of all members.](#)

## Public Types

```
enum EncodingType { k1X, k2X, k4X }
```

---

```
virtual void Start ()=0
virtual INT32 Get ()=0
virtual void Reset ()=0
virtual void Stop ()=0
virtual double GetPeriod ()=0
virtual void SetMaxPeriod (double maxPeriod)=0
virtual bool GetStopped ()=0
virtual bool GetDirection ()=0
```

---

## Detailed Description

Interface for counting the number of ticks on a digital input channel.

Encoders, Gear tooth sensors, and counters should all subclass this so it can be used to build more advanced classes for control and driving.

---

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

- C:/WindRiver/workspace/WPILib/**CounterBase.h**
- 

Generated by  1.7.2

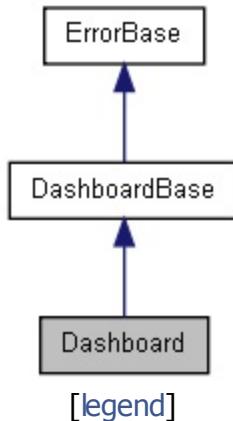


# Dashboard Class Reference

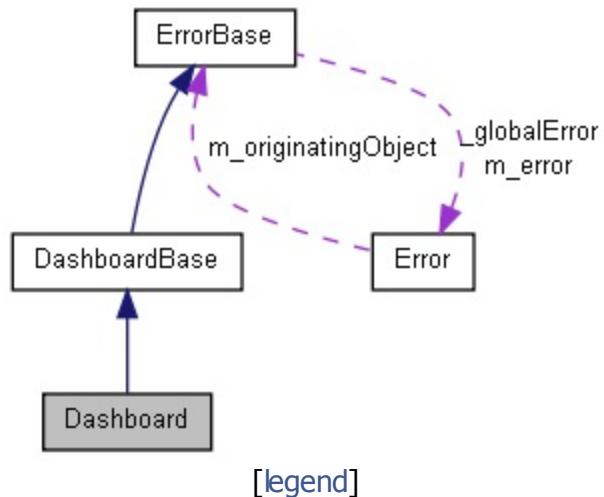
Pack data into the "user data" field that gets sent to the dashboard laptop via the driver station. [More...](#)

```
#include <Dashboard.h>
```

Inheritance diagram for Dashboard:



Collaboration diagram for Dashboard:



[List of all members.](#)

# Public Types

```
Type {  
    kI8, kI16, kI32, kU8,  
enum   kU16, kU32, kFloat, kDouble,  
        kBoolean, kString, kOther  
}  
enum  ComplexType { kArray, kCluster }
```

## **Dashboard** (SEM\_ID statusDataSemaphore)

**Dashboard** contructor.

virtual **~Dashboard** ()  
**Dashboard** destructor.

void **AddI8** (INT8 value)  
Pack a signed 8-bit int into the dashboard data structure.

void **AddI16** (INT16 value)  
Pack a signed 16-bit int into the dashboard data structure.

void **AddI32** (INT32 value)  
Pack a signed 32-bit int into the dashboard data structure.

void **AddU8** (UINT8 value)  
Pack an unsigned 8-bit int into the dashboard data structure.

void **AddU16** (UINT16 value)  
Pack an unsigned 16-bit int into the dashboard data structure.

void **AddU32** (UINT32 value)  
Pack an unsigned 32-bit int into the dashboard data structure.

void **AddFloat** (float value)  
Pack a 32-bit floating point number into the dashboard data structure.

void **AddDouble** (double value)  
Pack a 64-bit floating point number into the dashboard data structure.

void **AddBoolean** (bool value)  
Pack a boolean into the dashboard data structure.

void **AddString** (char \*value)  
Pack a NULL-terminated string of 8-bit characters into the dashboard data structure.

void **AddString** (char \*value, INT32 length)

Pack a string of 8-bit characters of specified length into the dashboard data structure.

**void AddArray ()**

Start an array in the packed dashboard data structure.

**void FinalizeArray ()**

Indicate the end of an array packed into the dashboard data structure.

**void AddCluster ()**

Start a cluster in the packed dashboard data structure.

**void FinalizeCluster ()**

Indicate the end of a cluster packed into the dashboard data structure.

**void Printf (const char \*writeFmt,...)**

Print a string to the UserData text on the **Dashboard**.

**INT32 Finalize ()**

Indicate that the packing is complete and commit the buffer to the **DriverStation**.

**void GetStatusBuffer (char \*\*userStatusData, INT32 \*userStatusDataSize)**

Called by the **DriverStation** class to retrieve buffers, sizes, etc.

**void Flush ()**

## Detailed Description

Pack data into the "user data" field that gets sent to the dashboard laptop via the driver station.

---

# Constructor & Destructor Documentation

## **Dashboard::Dashboard ( SEM\_ID statusDataSem ) [explicit]**

**Dashboard** constructor.

This is only called once when the **DriverStation** constructor is called.

## **Dashboard::~Dashboard ( ) [virtual]**

**Dashboard** destructor.

Called only when the **DriverStation** class is destroyed.

# Member Function Documentation

## **void Dashboard::AddArray( )**

Start an array in the packed dashboard data structure.

After calling [AddArray\(\)](#), call the appropriate Add method for each element of the array. Make sure you call the same add each time. An array must contain elements of the same type. You can use clusters inside of arrays to make each element of the array contain a structure of values. You can also nest arrays inside of other arrays. Every call to [AddArray\(\)](#) must have a matching call to [FinalizeArray\(\)](#).

## **void Dashboard::AddBoolean( bool value )**

Pack a boolean into the dashboard data structure.

### **Parameters:**

**value** Data to be packed into the structure.

## **void Dashboard::AddCluster( )**

Start a cluster in the packed dashboard data structure.

After calling [AddCluster\(\)](#), call the appropriate Add method for each element of the cluster. You can use clusters inside of arrays to make each element of the array contain a structure of values. Every call to [AddCluster\(\)](#) must have a matching call to [FinalizeCluster\(\)](#).

## **void Dashboard::AddDouble( double value )**

Pack a 64-bit floating point number into the dashboard data structure.

### **Parameters:**

**value** Data to be packed into the structure.

## **void Dashboard::AddFloat( float value )**

Pack a 32-bit floating point number into the dashboard data structure.

**Parameters:**

**value** Data to be packed into the structure.

**void Dashboard::AddI16 ( INT16 value )**

Pack a signed 16-bit int into the dashboard data structure.

**Parameters:**

**value** Data to be packed into the structure.

**void Dashboard::AddI32 ( INT32 value )**

Pack a signed 32-bit int into the dashboard data structure.

**Parameters:**

**value** Data to be packed into the structure.

**void Dashboard::AddI8 ( INT8 value )**

Pack a signed 8-bit int into the dashboard data structure.

**Parameters:**

**value** Data to be packed into the structure.

**void Dashboard::AddString ( char \* value )**

Pack a NULL-terminated string of 8-bit characters into the dashboard data structure.

**Parameters:**

**value** Data to be packed into the structure.

**void Dashboard::AddString ( char \* value,  
                          INT32 length  
                          )**

Pack a string of 8-bit characters of specified length into the dashboard data structure.

## Parameters:

**value** Data to be packed into the structure.

**length** The number of bytes in the string to pack.

### **void Dashboard::AddU16 ( **UINT16** value )**

Pack an unsigned 16-bit int into the dashboard data structure.

## Parameters:

**value** Data to be packed into the structure.

### **void Dashboard::AddU32 ( **UINT32** value )**

Pack an unsigned 32-bit int into the dashboard data structure.

## Parameters:

**value** Data to be packed into the structure.

### **void Dashboard::AddU8 ( **UINT8** value )**

Pack an unsigned 8-bit int into the dashboard data structure.

## Parameters:

**value** Data to be packed into the structure.

### **INT32 Dashboard::Finalize ( )**

Indicate that the packing is complete and commit the buffer to the **DriverStation**.

The packing of the dashboard packet is complete. If you are not using the packed dashboard data, you can call **Finalize()** to commit the **Printf()** buffer and the error string buffer. In effect, you are packing an empty structure. Prepares a packet to go to the dashboard...

## Returns:

The total size of the data packed into the userData field of the status packet.

## **void Dashboard::FinalizeArray( )**

Indicate the end of an array packed into the dashboard data structure.

After packing data into the array, call **FinalizeArray()**. Every call to **AddArray()** must have a matching call to **FinalizeArray()**.

## **void Dashboard::FinalizeCluster( )**

Indicate the end of a cluster packed into the dashboard data structure.

After packing data into the cluster, call **FinalizeCluster()**. Every call to **AddCluster()** must have a matching call to **FinalizeCluster()**.

## **void Dashboard::GetStatusBuffer ( char \*\* userStatusData,                                   INT32 \* userStatusDataSize                                   )**                          [**virtual**]

Called by the **DriverStation** class to retrieve buffers, sizes, etc.

for writing to the NetworkCommunication task. This function is called while holding the m\_statusDataSemaphore.

Implements **DashboardBase**.

## **void Dashboard::Printf ( const char \* writeFmt,                                   ...                                   )**

Print a string to the UserData text on the **Dashboard**.

This will add text to the buffer to send to the dashboard. You must call **Finalize()** periodically to actually send the buffer to the dashboard if you are not using the packed dashboard data.

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

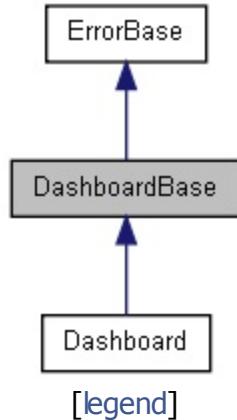
- C:/WindRiver/workspace/WPILib/**Dashboard.h**
- C:/WindRiver/workspace/WPILib/Dashboard.cpp



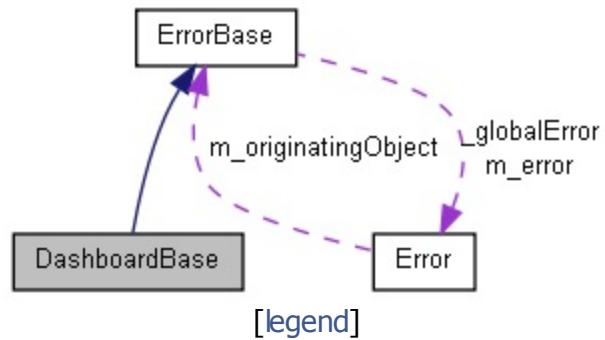


# DashboardBase Class Reference

Inheritance diagram for DashboardBase:



Collaboration diagram for DashboardBase:



List of all members.

## Public Member Functions

---

```
virtual void GetStatusBuffer (char **userStatusData, INT32  
*userStatusDataSize)=0  
virtual void Flush ()=0
```

---

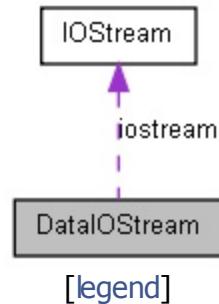
The documentation for this class was generated from the following file:

- C:/WindRiver/workspace/WPILib/**DashboardBase.h**



# DataIOStream Class Reference

Collaboration diagram for DataIOStream:



List of all members.

# Public Member Functions

<b>DataIOStream (IOStream *stream)</b>
void <b>writeByte</b> (uint8_t b)
void <b>write2BytesBE</b> (uint16_t s)
void <b>writeString</b> (std::string &str)
void <b>flush</b> ()
uint8_t <b>readByte</b> ()
uint16_t <b>read2BytesBE</b> ()
std::string * <b>readString</b> ()
void <b>close</b> ()

## IOStream & iostream

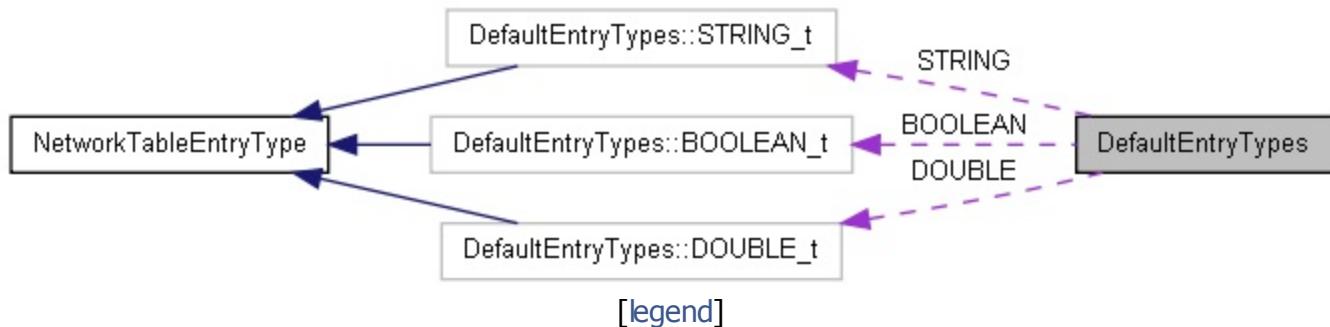
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/connection/**DataIOStream.h**
- C:/WindRiver/workspace/WPILib/networktables2/connection/DataIOStream.cpp



# DefaultEntryTypes Class Reference

Collaboration diagram for DefaultEntryTypes:



List of all members.

## Classes

class **BOOLEAN\_t**  
a boolean entry type

class **DOUBLE\_t**  
a double floating point entry type

class **STRING\_t**  
a string entry type

static void **registerTypes** ([NetworkTableEntryTypeManager](#)  
\*typeManager)

# Static Public Attributes

---

static BOOLEAN\_t **BOOLEAN**  
static DOUBLE\_t **DOUBLE**  
static STRING\_t **STRING**

---

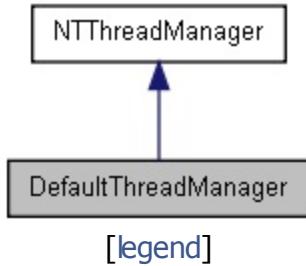
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/type/[DefaultEntryTypes.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/type/DefaultEntryTypes.cpp

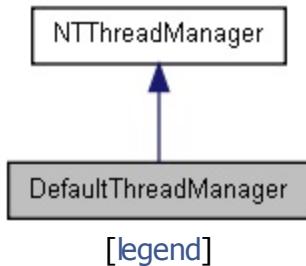
[Class List](#)[Class Hierarchy](#)[Class Members](#)

# DefaultThreadManager Class Reference

Inheritance diagram for DefaultThreadManager:



Collaboration diagram for DefaultThreadManager:



List of all members.

---

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

- C:/WindRiver/workspace/WPIlib/networktables2/thread/[DefaultThreadManager.h](#)
- C:/WindRiver/workspace/WPIlib/networktables2/thread/DefaultThreadManger.cpp

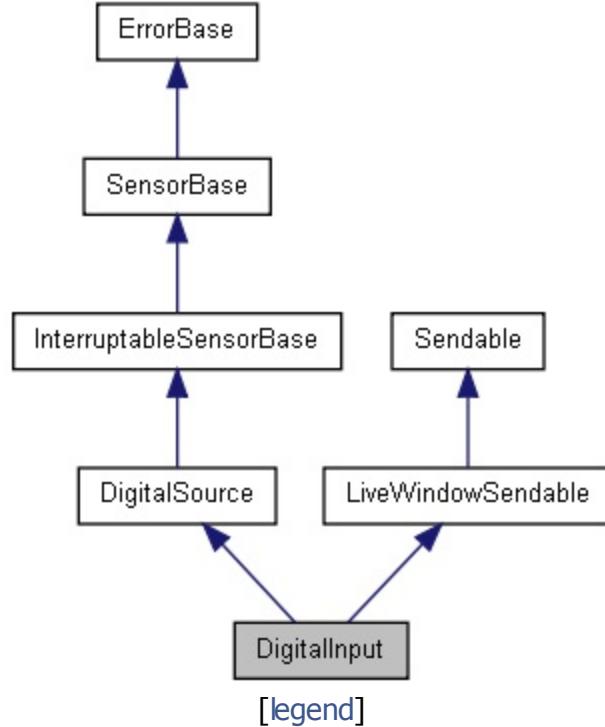


# DigitalInput Class Reference

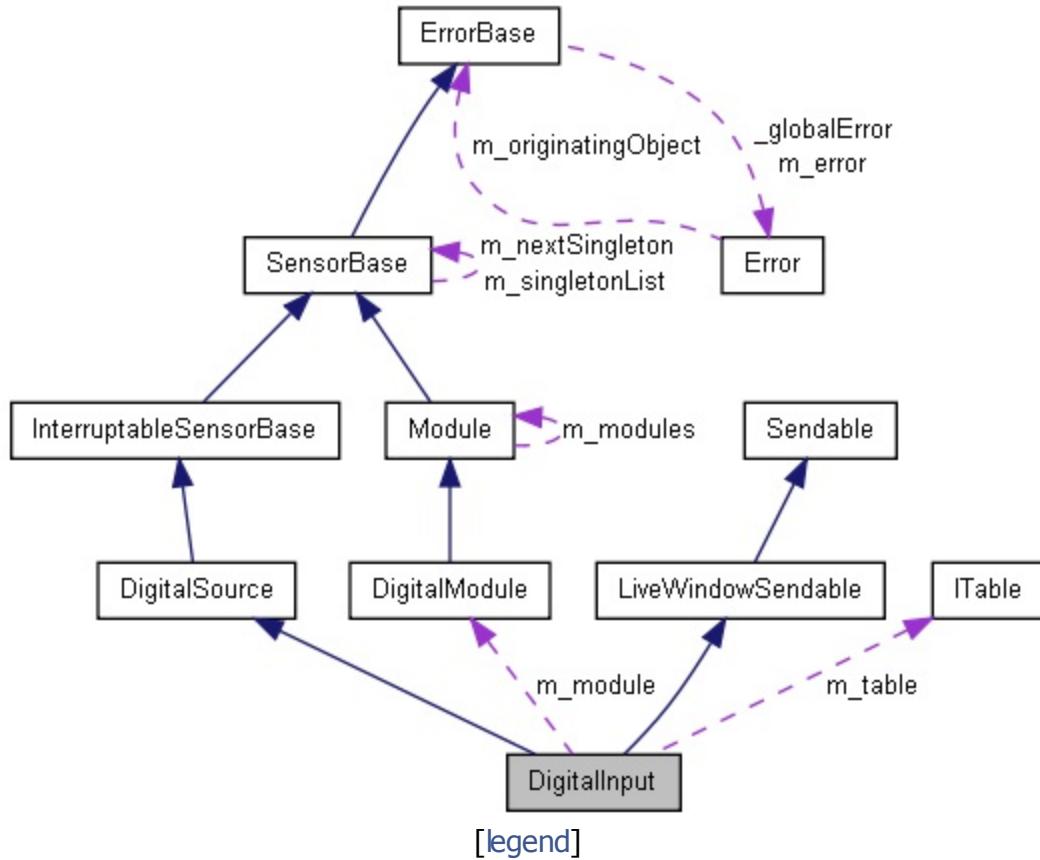
Class to read a digital input. [More...](#)

```
#include <DigitalInput.h>
```

Inheritance diagram for DigitalInput:



Collaboration diagram for DigitalInput:



List of all members.

# Public Member Functions

**DigitalInput** (UINT32 channel)

Create an instance of a Digital Input class.

**DigitalInput** (UINT8 moduleNumber, UINT32 channel)

Create an instance of a Digital Input class.

virtual **~DigitalInput** ()

Free resources associated with the Digital Input class.

UINT32 **Get** ()

UINT32 **GetChannel** ()

virtual UINT32 **GetChannelForRouting** ()

virtual UINT32 **GetModuleForRouting** ()

virtual bool **GetAnalogTriggerForRouting** ()

virtual void **RequestInterrupts** (tInterruptHandler handler, void \*param=NULL)  
Asynchronous handler version.

virtual void **RequestInterrupts** ()

Synchronous Wait version.

void **SetUpSourceEdge** (bool risingEdge, bool fallingEdge)

void **UpdateTable** ()

Update the table for this sendable object with the latest values.

void **StartLiveWindowMode** ()

Start having this sendable object automatically respond to value changes reflect the value on the table.

void **StopLiveWindowMode** ()

Stop having this sendable object automatically respond to value changes.

std::string **GetSmartDashboardType** ()

void **InitTable** (ITable \*subTable)

Initializes a table for this sendable object.

**ITable** \* **GetTable** ()

## Detailed Description

Class to read a digital input.

This class will read digital inputs and return the current value on the channel. Other devices such as encoders, gear tooth sensors, etc. that are implemented elsewhere will automatically allocate digital inputs and outputs as required. This class is only for devices like switches etc. that aren't implemented anywhere else.

---

# Constructor & Destructor Documentation

## DigitalInput::DigitalInput( **UINT32 channel** ) [explicit]

Create an instance of a Digital Input class.

Creates a digital input given a channel and uses the default module.

### Parameters:

**channel** The digital channel (1..14).

## DigitalInput::DigitalInput( **UINT8 moduleNumber,** **UINT32 channel** )

Create an instance of a Digital Input class.

Creates a digital input given an channel and module.

### Parameters:

**moduleNumber** The digital module (1 or 2).

**channel** The digital channel (1..14).

# Member Function Documentation

**bool DigitalInput::GetAnalogTriggerForRouting( ) [virtual]**

## Returns:

The value to be written to the analog trigger field of a routing mux.

Implements **DigitalSource**.

**UINT32 DigitalInput::GetChannel( )**

## Returns:

The GPIO channel number that this object represents.

**UINT32 DigitalInput::GetChannelForRouting( ) [virtual]**

## Returns:

The value to be written to the channel field of a routing mux.

Implements **DigitalSource**.

**UINT32 DigitalInput::GetModuleForRouting( ) [virtual]**

## Returns:

The value to be written to the module field of a routing mux.

Implements **DigitalSource**.

**std::string DigitalInput::GetSmartDashboardType( ) [virtual]**

## Returns:

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

**ITable \* DigitalInput::GetTable( ) [virtual]**

**Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

**void DigitalInput::InitTable ( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

**Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

**void DigitalInput::RequestInterrupts ( ) [virtual]**

Synchronous Wait version.

Request interrupts synchronously on this digital input.

Request interrupts in synchronous mode where the user program will have to explicitly wait for the interrupt to occur. The default is interrupt on rising edges only.

Implements **DigitalSource**.

**void DigitalInput::RequestInterrupts ( tInterruptHandler handler,  
void \* param = NULL [virtual] )**

Asynchronous handler version.

Request interrupts asynchronously on this digital input.

**Parameters:**

**handler** The address of the interrupt handler function of type **tInterruptHandler** that will be called whenever there is an interrupt on the digital input port. Request interrupts in synchronous mode where the user program interrupt handler will be called when an interrupt occurs. The default is interrupt on rising edges only.

Implements **DigitalSource**.

---

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

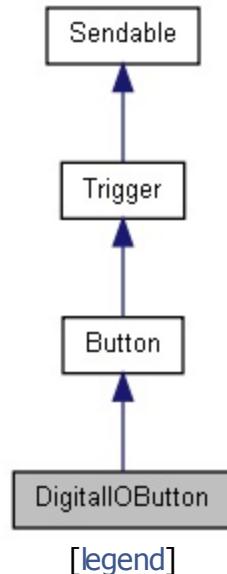
- C:/WindRiver/workspace/WPILib/**DigitalInput.h**
  - C:/WindRiver/workspace/WPILib/DigitalInput.cpp
- 

Generated by [doxygen](#) 1.7.2



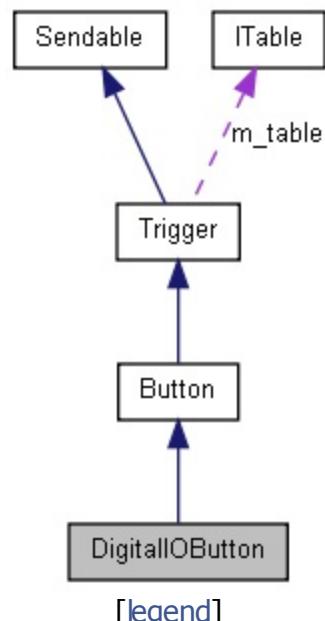
# DigitalIOButton Class Reference

Inheritance diagram for DigitalIOButton:



[legend]

Collaboration diagram for DigitalIOButton:



[legend]

List of all members.

# Public Member Functions

**DigitalIOButton** (int port)

virtual bool **Get** ()

static const bool **kActiveState** = false

---

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

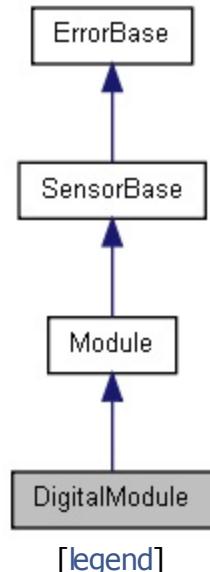
- C:/WindRiver/workspace/WPILib/Buttons/**DigitalIOButton.h**
- C:/WindRiver/workspace/WPILib/Buttons/DigitalIOButton.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

[Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Protected Member Functions](#) | [Friends](#)

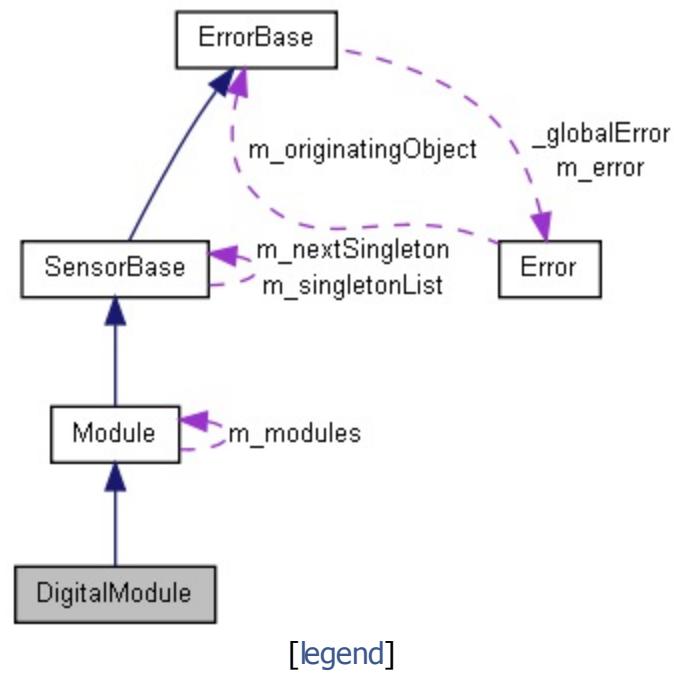
# DigitalModule Class Reference

Inheritance diagram for DigitalModule:



[legend]

Collaboration diagram for DigitalModule:



[legend]

List of all members.

# Public Member Functions

void	<b>SetPWM</b> (UINT32 channel, UINT8 value) Set a <b>PWM</b> channel to the desired value.
UINT8	<b>GetPWM</b> (UINT32 channel) Get a value from a <b>PWM</b> channel.
void	<b>SetPWMPPeriodScale</b> (UINT32 channel, UINT32 squelchMask) Set how often the <b>PWM</b> signal is squelched, thus scaling the period.
void	<b>SetRelayForward</b> (UINT32 channel, bool on) Set the state of a relay.
void	<b>SetRelayReverse</b> (UINT32 channel, bool on) Set the state of a relay.
bool	<b>GetRelayForward</b> (UINT32 channel) Get the current state of the forward relay channel.
UINT8	<b>GetRelayForward</b> () Get the current state of all of the forward relay channels on this module.
bool	<b>GetRelayReverse</b> (UINT32 channel) Get the current state of the reverse relay channel.
UINT8	<b>GetRelayReverse</b> () Get the current state of all of the reverse relay channels on this module.
bool	<b>AllocateDIO</b> (UINT32 channel, bool input) Allocate Digital I/O channels.
void	<b>FreeDIO</b> (UINT32 channel) Free the resource associated with a digital I/O channel.
void	<b>SetDIO</b> (UINT32 channel, short value) Write a digital I/O bit to the FPGA.
bool	<b>GetDIO</b> (UINT32 channel) Read a digital I/O bit from the FPGA.
UINT16	<b>GetDIO</b> () Read the state of all the Digital I/O lines from the FPGA These are not remapped to logical order.
bool	<b>GetDIODirection</b> (UINT32 channel) Read the direction of a the Digital I/O lines A 1 bit means output and a 0 bit means input.
UINT16	<b>GetDIODirection</b> ()

Read the direction of all the Digital I/O lines from the FPGA  
A 1 bit means output and a 0 bit means input.

**void** **Pulse** (UINT32 channel, float pulseLength)

Generate a single pulse.

**bool** **IsPulsing** (UINT32 channel)

Check a DIO line to see if it is currently generating a pulse.

**bool** **IsPulsing** ()

Check if any DIO line is currently generating a pulse.

**UINT32** **AllocateDO\_PWM** ()

Allocate a DO **PWM** Generator.

**void** **FreeDO\_PWM** (UINT32 pwmGenerator)

Free the resource associated with a DO **PWM** generator.

**void** **SetDO\_PWMRate** (float rate)

Change the frequency of the DO **PWM** generator.

**void** **SetDO\_PWM\_DutyCycle** (UINT32 pwmGenerator, float dutyCycle)

Configure the duty-cycle of the **PWM** generator.

**void** **SetDO\_PWM\_OutputChannel** (UINT32 pwmGenerator, UINT32 channel)

Configure which DO channel the **PWM** signal is output on.

**I2C \*** **GetI2C** (UINT32 address)

Return a pointer to an **I2C** object for this digital module. The caller is responsible for deleting the pointer.

# Static Public Member Functions

static **DigitalModule** \* **GetInstance** (UINT8 moduleNumber)

Get an instance of an Digital **Module**.

static UINT8 **RemapDigitalChannel** (UINT32 channel)

static UINT8 **UnmapDigitalChannel** (UINT32 channel)

**DigitalModule** (UINT8 moduleNumber)

Create a new instance of an digital module.

# **Friends**

class **I2C**  
class **Module**

---

# Constructor & Destructor Documentation

## DigitalModule::DigitalModule ( **UINT8 moduleNumber** ) [explicit, protected]

Create a new instance of an digital module.

Create an instance of the digital module object. Initialize all the parameters to reasonable values on start. Setting a global value on an digital module can be done only once unless subsequent values are set the previously set value. Digital modules are a singleton, so the constructor is never called outside of this class.

### Parameters:

**moduleNumber** The digital module to create (1 or 2).

# Member Function Documentation

```
bool DigitalModule::AllocateDIO( UINT32 channel,  
                                bool      input  
                            )
```

Allocate Digital I/O channels.

Allocate channels so that they are not accidentally reused. Also the direction is set at the time of the allocation.

## Parameters:

**channel** The Digital I/O channel  
**input** If true open as input; if false open as output

## Returns:

Was successfully allocated

## UINT32 DigitalModule::AllocateDO\_PWM( )

Allocate a DO **PWM** Generator.

Allocate **PWM** generators so that they are not accidentally reused.

## Returns:

**PWM** Generator refnum

## void DigitalModule::FreeDIO( UINT32 channel )

Free the resource associated with a digital I/O channel.

## Parameters:

**channel** The Digital I/O channel to free

## void DigitalModule::FreeDO\_PWM( UINT32 pwmGenerator )

Free the resource associated with a DO **PWM** generator.

## Parameters:

**pwmGenerator** The pwmGen to free that was allocated with  
[AllocateDO\\_PWM\(\)](#)

## **bool DigitalModule::GetDIO ( UINT32 channel )**

Read a digital I/O bit from the FPGA.

Get a single value from a digital I/O channel.

### **Parameters:**

**channel** The digital I/O channel

### **Returns:**

The state of the specified channel

## **UINT16 DigitalModule::GetDIO ( )**

Read the state of all the Digital I/O lines from the FPGA These are not remapped to logical order.

They are still in hardware order.

## **UINT16 DigitalModule::GetDIODirection ( )**

Read the direction of all the Digital I/O lines from the FPGA A 1 bit means output and a 0 bit means input.

These are not remapped to logical order. They are still in hardware order.

## **bool DigitalModule::GetDIODirection ( UINT32 channel )**

Read the direction of a the Digital I/O lines A 1 bit means output and a 0 bit means input.

### **Parameters:**

**channel** The digital I/O channel

### **Returns:**

The direction of the specified channel

## **I2C \* DigitalModule::GetI2C( UINT32 address )**

Return a pointer to an **I2C** object for this digital module. The caller is responsible for deleting the pointer.

### **Parameters:**

**address** The address of the device on the **I2C** bus

### **Returns:**

A pointer to an **I2C** object to talk to the device at address

## **DigitalModule \* DigitalModule::GetInstance( UINT8 moduleNumber ) [static]**

Get an instance of an **Digital Module**.

Singleton digital module creation where a module is allocated on the first use and the same module is returned on subsequent uses.

### **Parameters:**

**moduleNumber** The digital module to get (1 or 2).

## **UINT8 DigitalModule::GetPWM( UINT32 channel )**

Get a value from a **PWM** channel.

The values range from 0 to 255.

### **Parameters:**

**channel** The **PWM** channel to read from.

### **Returns:**

The raw **PWM** value.

## **bool DigitalModule::IsPulsing( )**

Check if any DIO line is currently generating a pulse.

### **Returns:**

A pulse on some line is in progress

## **bool DigitalModule::IsPulsing ( UINT32 channel )**

Check a DIO line to see if it is currently generating a pulse.

### **Returns:**

A pulse is in progress

## **void DigitalModule::Pulse ( UINT32 channel, float pulseLength )**

Generate a single pulse.

Write a pulse to the specified digital output channel. There can only be a single pulse going at any time.

### **Parameters:**

**channel** The Digital Output channel that the pulse should be output on  
**pulseLength** The active length of the pulse (in seconds)

## **void DigitalModule::SetDIO ( UINT32 channel, short value )**

Write a digital I/O bit to the FPGA.

Set a single value on a digital I/O channel.

### **Parameters:**

**channel** The Digital I/O channel  
**value** The state to set the digital channel (if it is configured as an output)

## **void DigitalModule::SetDO\_PWM\_DutyCycle ( UINT32 pwmGenerator, float dutyCycle )**

Configure the duty-cycle of the **PWM** generator.

## Parameters:

**pwmGenerator** The generator index reserved by [AllocateDO\\_PWM\(\)](#)

**dutyCycle** The percent duty cycle to output [0..1].

```
void DigitalModule::SetDO_PWMOutputChannel ( UINT32 pwmGenerator,  
                                            UINT32 channel  
)
```

Configure which DO channel the **PWM** signal is output on.

## Parameters:

**pwmGenerator** The generator index reserved by [AllocateDO\\_PWM\(\)](#)

**channel** The Digital Output channel to output on

```
void DigitalModule::SetDO_PWMRate ( float rate )
```

Change the frequency of the DO **PWM** generator.

The valid range is from 0.6 Hz to 19 kHz. The frequency resolution is logarithmic.

## Parameters:

**rate** The frequency to output all digital output **PWM** signals on this module.

```
void DigitalModule::SetPWM ( UINT32 channel,  
                            UINT8 value  
)
```

Set a **PWM** channel to the desired value.

The values range from 0 to 255 and the period is controlled by the **PWM** Period and MinHigh registers.

## Parameters:

**channel** The **PWM** channel to set.

**value** The **PWM** value to set.

```
void DigitalModule::SetPWMPulseScale ( UINT32 channel,
```

**UINT32 squelchMask**

)

Set how often the **PWM** signal is squelched, thus scaling the period.

**Parameters:**

**channel** The **PWM** channel to configure.  
**squelchMask** The 2-bit mask of outputs to squelch.

**void DigitalModule::SetRelayForward ( **UINT32** channel,  
                                  **bool**     **on** )**

Set the state of a relay.

Set the state of a relay output to be forward. Relays have two outputs and each is independently set to 0v or 12v.

**void DigitalModule::SetRelayReverse ( **UINT32** channel,  
                                  **bool**     **on** )**

Set the state of a relay.

Set the state of a relay output to be reverse. Relays have two outputs and each is independently set to 0v or 12v.

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

- C:/WindRiver/workspace/WPIlib/**DigitalModule.h**
- C:/WindRiver/workspace/WPIlib/DigitalModule.cpp

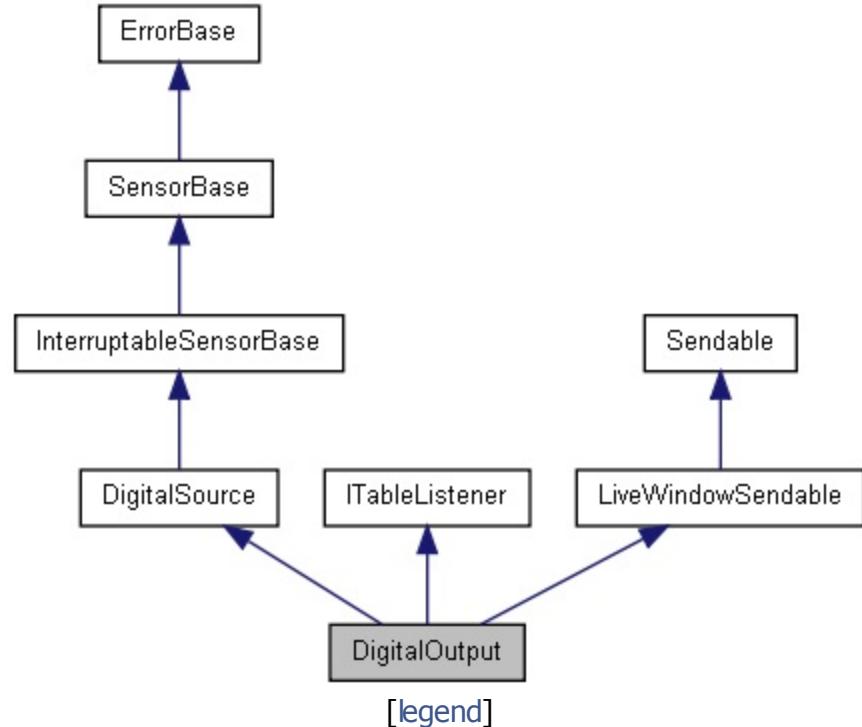


# DigitalOutput Class Reference

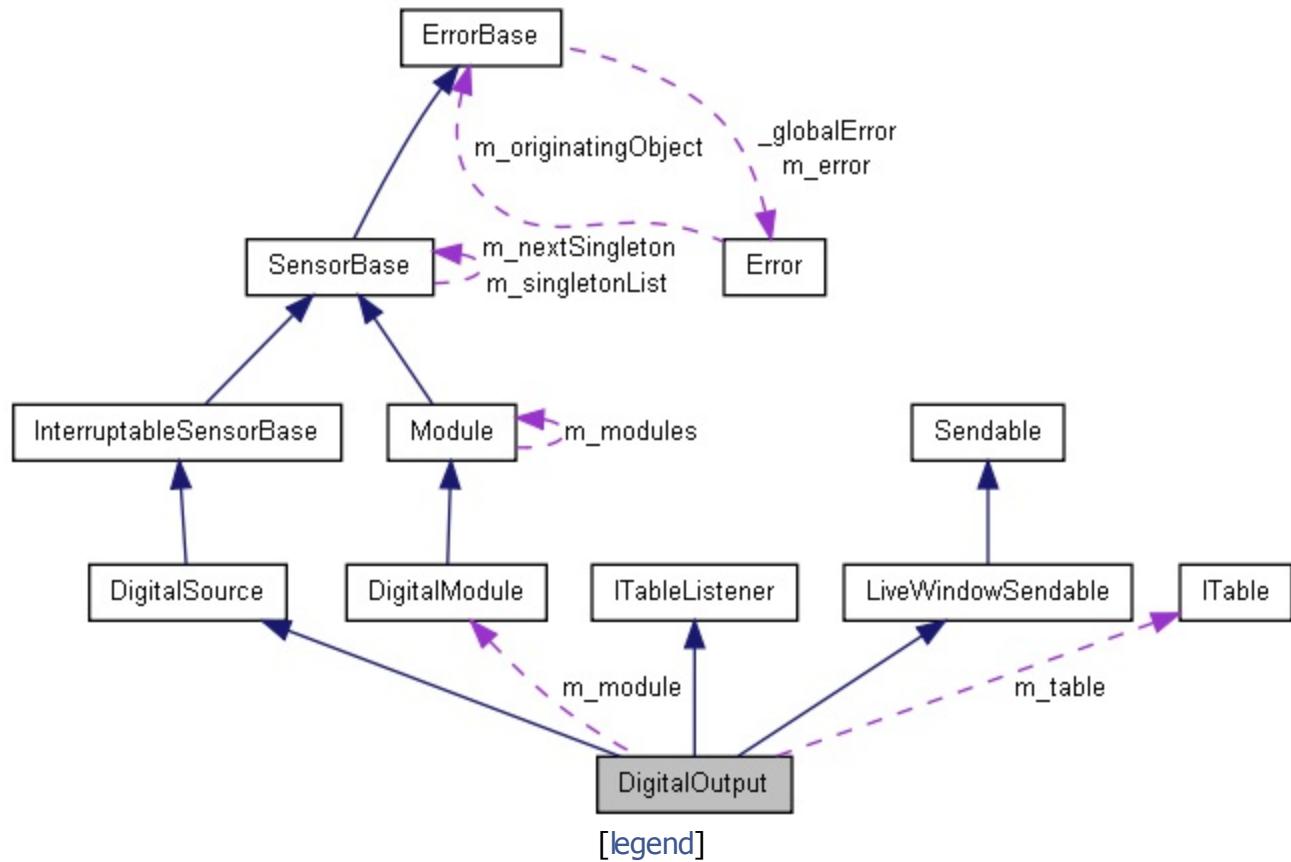
Class to write to digital outputs. [More...](#)

```
#include <DigitalOutput.h>
```

Inheritance diagram for DigitalOutput:



Collaboration diagram for DigitalOutput:



List of all members.

# Public Member Functions

**DigitalOutput** (UINT32 channel)

Create an instance of a digital output.

**DigitalOutput** (UINT8 moduleNumber, UINT32 channel)

Create an instance of a digital output.

**virtual ~DigitalOutput ()**

Free the resources associated with a digital output.

**void Set (UINT32 value)**

Set the value of a digital output.

**UINT32 GetChannel ()**

**void Pulse (float length)**

Output a single pulse on the digital output line.

**bool IsPulsing ()**

Determine if the pulse is still going.

**void SetPWMRate (float rate)**

Change the **PWM** frequency of the **PWM** output on a Digital Output line.

**void EnablePWM (float initialDutyCycle)**

Enable a **PWM** Output on this line.

**void DisablePWM ()**

Change this line from a **PWM** output back to a static Digital Output line.

**void UpdateDutyCycle (float dutyCycle)**

Change the duty-cycle that is being generated on the line.

**virtual UINT32 GetChannelForRouting ()**

**virtual UINT32 GetModuleForRouting ()**

**virtual bool GetAnalogTriggerForRouting ()**

**virtual void RequestInterrupts (tInterruptHandler handler, void \*param)**  
Request interrupts asynchronously on this digital output.

**virtual void RequestInterrupts ()**

Request interrupts synchronously on this digital output.

**void SetUpSourceEdge (bool risingEdge, bool fallingEdge)**

**virtual void ValueChanged (ITable \*source, const std::string &key, EntryValue value, bool isNew)**

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

**void UpdateTable ()**

	Update the table for this sendable object with the latest values.
void <b>StartLiveWindowMode ()</b>	Start having this sendable object automatically respond to value changes reflect the value on the table.
void <b>StopLiveWindowMode ()</b>	Stop having this sendable object automatically respond to value changes.
std::string <b>GetSmartDashboardType ()</b>	
void <b>InitTable (ITable *subTable)</b>	Initializes a table for this sendable object.
<b>ITable * GetTable ()</b>	

## Detailed Description

Class to write to digital outputs.

Write values to the digital output channels. Other devices implemented elsewhere will allocate channels automatically so for those devices it shouldn't be done here.

---

# Constructor & Destructor Documentation

## DigitalOutput::DigitalOutput( **UINT32 channel** ) [explicit]

Create an instance of a digital output.

Create a digital output given a channel. The default module is used.

### Parameters:

**channel** The digital channel (1..14).

## DigitalOutput::DigitalOutput( **UINT8 moduleNumber,** **UINT32 channel** )

Create an instance of a digital output.

Create an instance of a digital output given a module number and channel.

### Parameters:

**moduleNumber** The digital module (1 or 2).

**channel** The digital channel (1..14).

# Member Function Documentation

## **void DigitalOutput::DisablePWM ( )**

Change this line from a **PWM** output back to a static Digital Output line.

Free up one of the 4 DO **PWM** generator resources that were in use.

## **void DigitalOutput::EnablePWM ( float initialDutyCycle )**

Enable a **PWM** Output on this line.

Allocate one of the 4 DO **PWM** generator resources from this module.

Supply the initial duty-cycle to output so as to avoid a glitch when first starting.

The resolution of the duty cycle is 8-bit for low frequencies (1kHz or less) but is reduced the higher the frequency of the **PWM** signal is.

### **Parameters:**

**initialDutyCycle** The duty-cycle to start generating. [0..1]

## **bool DigitalOutput::GetAnalogTriggerForRouting ( ) [virtual]**

### **Returns:**

The value to be written to the analog trigger field of a routing mux.

Implements **DigitalSource**.

## **UINT32 DigitalOutput::GetChannel ( )**

### **Returns:**

The GPIO channel number that this object represents.

## **UINT32 DigitalOutput::GetChannelForRouting ( ) [virtual]**

### **Returns:**

The value to be written to the channel field of a routing mux.

Implements **DigitalSource**.

## **UINT32 DigitalOutput::GetModuleForRouting ( ) [virtual]**

### **Returns:**

The value to be written to the module field of a routing mux.

Implements **DigitalSource**.

## **std::string DigitalOutput::GetSmartDashboardType ( ) [virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

## **ITable \* DigitalOutput::GetTable ( ) [virtual]**

### **Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

## **void DigitalOutput::InitTable ( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

## **bool DigitalOutput::IsPulsing ( )**

Determine if the pulse is still going.

Determine if a previously started pulse is still going.

## **void DigitalOutput::Pulse ( float length )**

Output a single pulse on the digital output line.

Send a single pulse on the digital output line where the pulse diration is specified in seconds. Maximum pulse length is 0.0016 seconds.

### **Parameters:**

**length** The pulselength in seconds

## **void DigitalOutput::RequestInterrupts ( ) [virtual]**

Request interrupts synchronously on this digital output.

Request interrupts in synchronus mode where the user program will have to explicitly wait for the interrupt to occur. The default is interrupt on rising edges only.

Implements **DigitalSource**.

## **void DigitalOutput::RequestInterrupts ( tInterruptHandler handler, void \* param ) [virtual]**

Request interrupts asynchronously on this digital output.

### **Parameters:**

**handler** The address of the interrupt handler function of type tInterruptHandler that will be called whenever there is an interrupt on the digital output port. Request interrupts in synchronus mode where the user program interrupt handler will be called when an interrupt occurs. The default is interrupt on rising edges only.

Implements **DigitalSource**.

## **void DigitalOutput::Set ( UINT32 value )**

Set the value of a digital output.

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

## **void DigitalOutput::SetPWMRate ( float rate )**

Change the **PWM** frequency of the **PWM** output on a Digital Output line.

The valid range is from 0.6 Hz to 19 kHz. The frequency resolution is logarithmic.

There is only one **PWM** frequency per digital module.

### **Parameters:**

**rate** The frequency to output all digital output **PWM** signals on this module.

## **void DigitalOutput::UpdateDutyCycle ( float dutyCycle )**

Change the duty-cycle that is being generated on the line.

The resolution of the duty cycle is 8-bit for low frequencies (1kHz or less) but is reduced the higher the frequency of the **PWM** signal is.

### **Parameters:**

**dutyCycle** The duty-cycle to change to. [0..1]

## **void DigitalOutput::ValueChanged ( ITable \* source, const std::string & key, EntryValue value, bool isNew ) [virtual]**

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

### **Parameters:**

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

---

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

- C:/WindRiver/workspace/WPIlib/**DigitalOutput.h**
  - C:/WindRiver/workspace/WPIlib/DigitalOutput.cpp
- 

Generated by  1.7.2

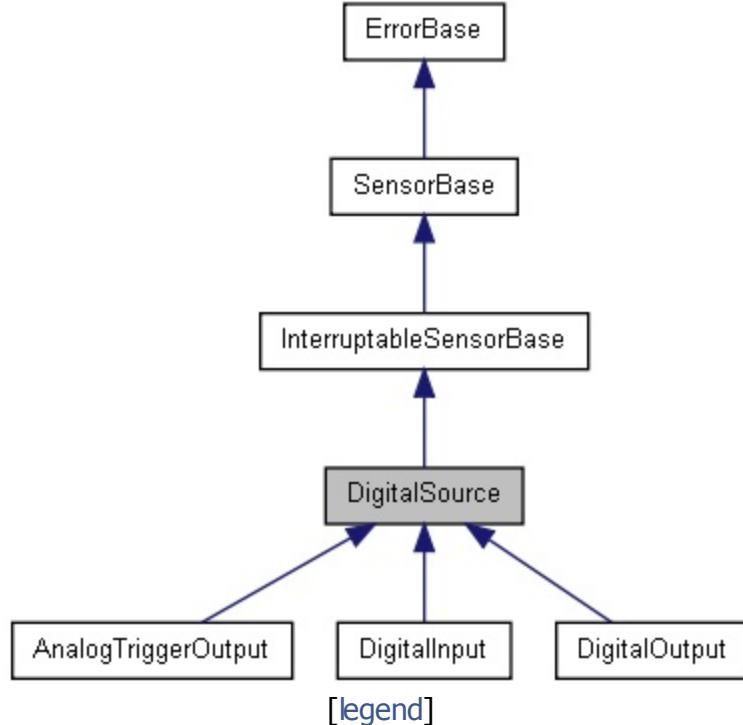


# DigitalSource Class Reference

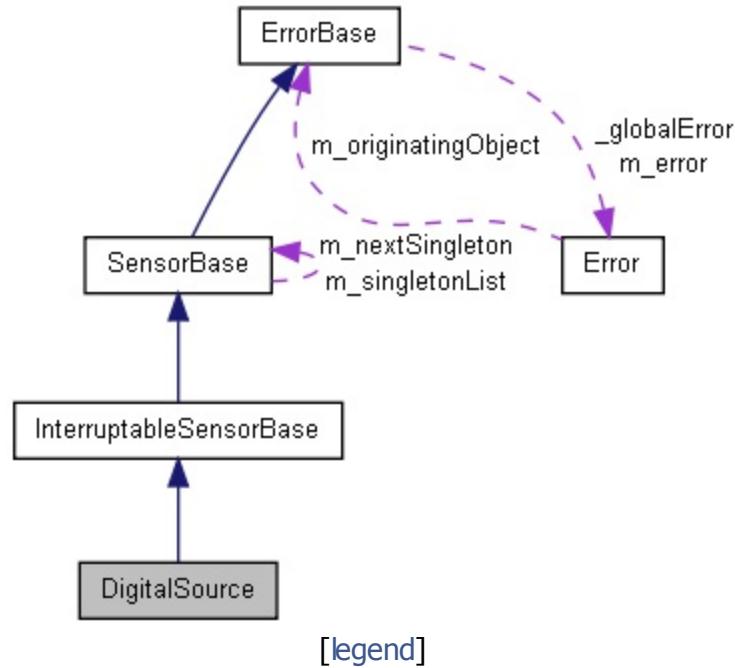
## DigitalSource Interface. More...

```
#include <DigitalSource.h>
```

Inheritance diagram for DigitalSource:



Collaboration diagram for DigitalSource:



List of all members.

## Public Member Functions

virtual ~**DigitalSource** ()

**DigitalSource** destructor.

virtual UINT32 **GetChannelForRouting** ()=0

virtual UINT32 **GetModuleForRouting** ()=0

virtual bool **GetAnalogTriggerForRouting** ()=0

virtual void **RequestInterrupts** (tInterruptHandler handler, void \*param)=0  
Asynchronous handler version.

virtual void **RequestInterrupts** ()=0

Synchronous Wait version.

# Detailed Description

## DigitalSource Interface.

The **DigitalSource** represents all the possible inputs for a counter or a quadrature encoder. The source may be either a digital input or an analog input. If the caller just provides a channel, then a digital input will be constructed and freed when finished for the source. The source can either be a digital input or analog trigger but not both.

---

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

- C:/WindRiver/workspace/WPIlib/**DigitalSource.h**
- C:/WindRiver/workspace/WPIlib/DigitalSource.cpp

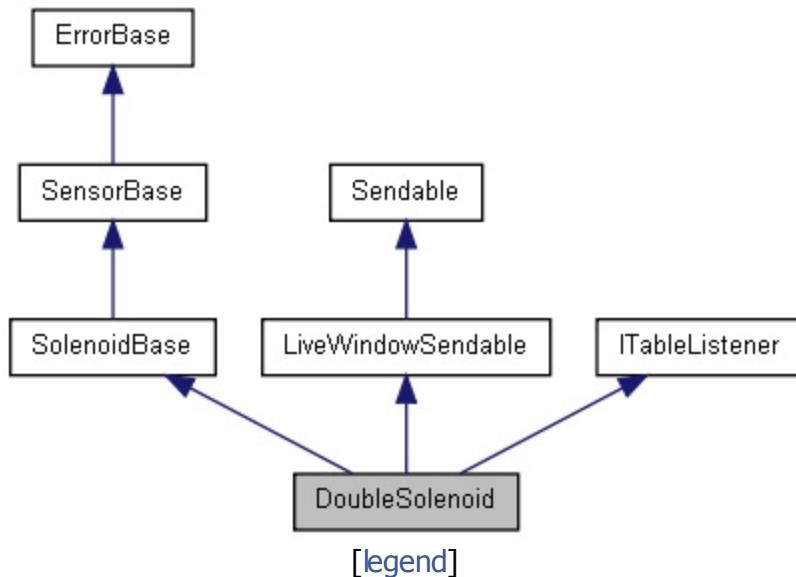


# DoubleSolenoid Class Reference

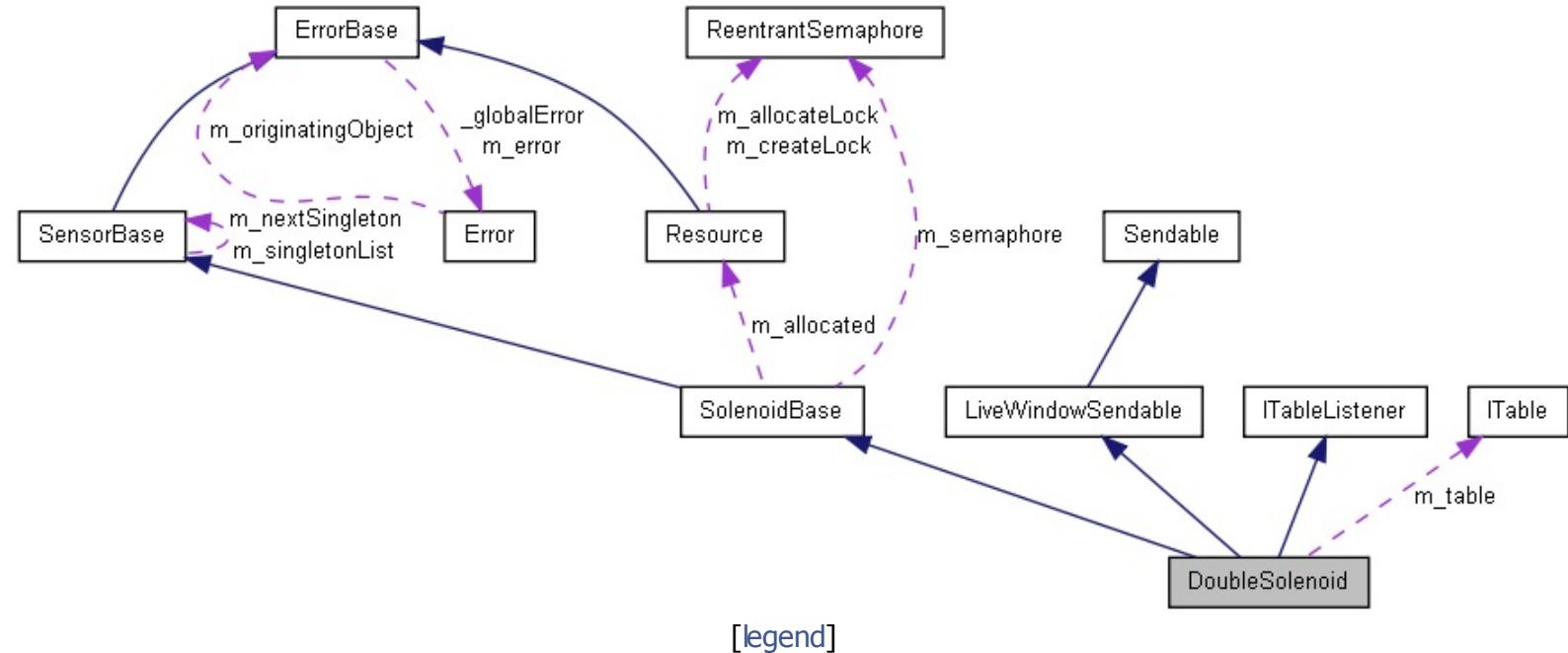
**DoubleSolenoid** class for running 2 channels of high voltage Digital Output (9472 module). [More...](#)

```
#include <DoubleSolenoid.h>
```

Inheritance diagram for DoubleSolenoid:



Collaboration diagram for DoubleSolenoid:



List of all members.

## Public Types

enum **Value** { **kOff**, **kForward**, **kReverse** }

**DoubleSolenoid** (UINT32 forwardChannel, UINT32 reverseChannel)

Constructor.

**DoubleSolenoid** (UINT8 moduleNumber, UINT32 forwardChannel, UINT32 reverseChannel)

Constructor.

virtual **~DoubleSolenoid** ()

Destructor.

virtual void **Set** (Value value)

Set the value of a solenoid.

virtual Value **Get** ()

Read the current value of the solenoid.

void **ValueChanged** (ITable \*source, const std::string &key, **EntryValue** value, bool isNew)

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

void **UpdateTable** ()

Update the table for this sendable object with the latest values.

void **StartLiveWindowMode** ()

Start having this sendable object automatically respond to value changes reflect the value on the table.

void **StopLiveWindowMode** ()

Stop having this sendable object automatically respond to value changes.

std::string **GetSmartDashboardType** ()

void **InitTable** (ITable \*subTable)

Initializes a table for this sendable object.

**ITable** \* **GetTable** ()

## Detailed Description

**DoubleSolenoid** class for running 2 channels of high voltage Digital Output (9472 module).

The **DoubleSolenoid** class is typically used for pneumatics solenoids that have two positions controlled by two separate channels.

---

# Constructor & Destructor Documentation

```
DoubleSolenoid::DoubleSolenoid ( UINT32 forwardChannel,  
                                UINT32 reverseChannel  
                                )  
                                [explicit]
```

Constructor.

## Parameters:

**forwardChannel** The forward channel on the module to control.  
**reverseChannel** The reverse channel on the module to control.

```
DoubleSolenoid::DoubleSolenoid ( UINT8 moduleNumber,  
                                UINT32 forwardChannel,  
                                UINT32 reverseChannel  
                                )
```

Constructor.

## Parameters:

**moduleNumber** The solenoid module (1 or 2).  
**forwardChannel** The forward channel on the module to control.  
**reverseChannel** The reverse channel on the module to control.

# Member Function Documentation

## **DoubleSolenoid::Value DoubleSolenoid::Get( ) [virtual]**

Read the current value of the solenoid.

### **Returns:**

The current value of the solenoid.

## **std::string DoubleSolenoid::GetSmartDashboardType( ) [virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

## **ITable \* DoubleSolenoid::GetTable( ) [virtual]**

### **Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

## **void DoubleSolenoid::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

## **void DoubleSolenoid::Set( Value value ) [virtual]**

Set the value of a solenoid.

### **Parameters:**

**value** Move the solenoid to forward, reverse, or don't move it.

```
void DoubleSolenoid::ValueChanged ( ITable * source,  
                                    const std::string & key,  
                                    EntryValue value,  
                                    bool isNew  
                                ) [virtual]
```

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

#### Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

---

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

- C:/WindRiver/workspace/WPILib/**DoubleSolenoid.h**
- C:/WindRiver/workspace/WPILib/DoubleSolenoid.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

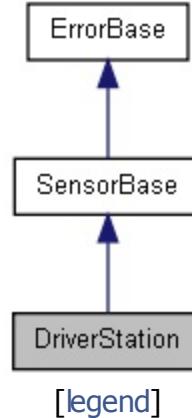
[Public Types](#) | [Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Static Public Attributes](#) |  
[Protected Member Functions](#)

# DriverStation Class Reference

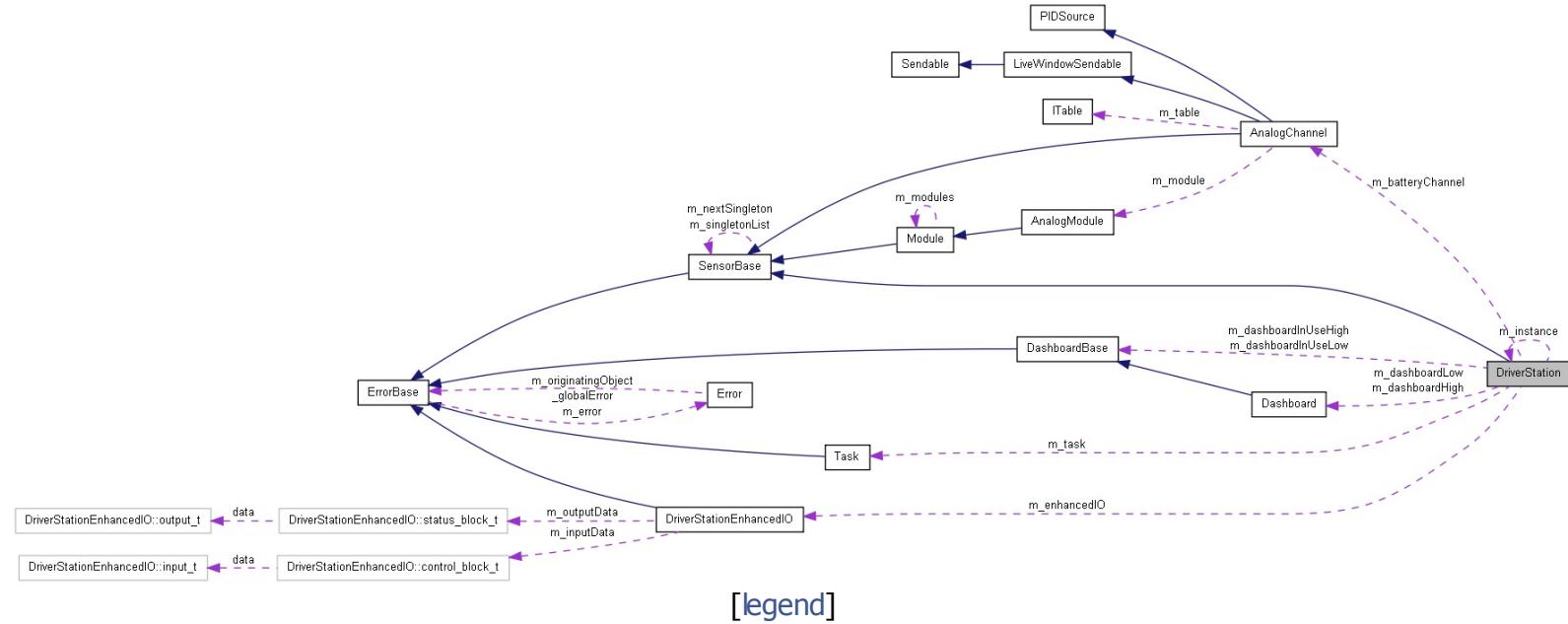
Provide access to the network communication data to / from the Driver Station. More...

```
#include <DriverStation.h>
```

Inheritance diagram for DriverStation:



Collaboration diagram for DriverStation:



List of all members.

# Public Types

```
enum Alliance { kRed, kBlue, kInvalid }
```

float	<b>GetStickAxis</b> (UINT32 stick, UINT32 axis) Get the value of the axis on a joystick.
short	<b>GetStickButtons</b> (UINT32 stick) The state of the buttons on the joystick.
float	<b>GetAnalogIn</b> (UINT32 channel) Get an analog voltage from the Driver Station.
bool	<b>GetDigitalIn</b> (UINT32 channel) Get values from the digital inputs on the Driver Station.
void	<b>SetDigitalOut</b> (UINT32 channel, bool value) Set a value for the digital outputs on the Driver Station.
bool	<b>GetDigitalOut</b> (UINT32 channel) Get a value that was set for the digital outputs on the Driver Station.
bool	<b>IsEnabled</b> ()
bool	<b>IsDisabled</b> ()
bool	<b>IsAutonomous</b> ()
bool	<b>IsOperatorControl</b> ()
bool	<b>IsTest</b> ()
bool	<b>IsNewControlData</b> () Has a new control packet from the driver station arrived since the last time this function was called? Warning: If you call this function from more than one place at the same time, you will not get the get the

intended behavior.

bool **IsFMSAttached ()**

Is the driver station attached to a Field Management System? Note: This does not work with the Blue DS.

UINT32 **GetPacketNumber ()**

Return the DS packet number.

Alliance **GetAlliance ()**

Return the alliance that the driver station says it is on.

UINT32 **GetLocation ()**

Return the driver station location on the field This could return 1, 2, or 3.

void **WaitForData ()**

Wait until a new packet comes from the driver station This blocks on a semaphore, so the waiting is efficient.

double **GetMatchTime ()**

Return the approximate match time The FMS does not currently send the official match time to the robots This returns the time since the enable signal sent from the Driver Station At the beginning of autonomous, the time is reset to 0.0 seconds At the beginning of teleop, the time is reset to +15.0 seconds If the robot is disabled, this returns 0.0 seconds Warning: This is not an official time (so it cannot be used to argue with referees)

float **GetBatteryVoltage ()**

Read the battery voltage from the specified **AnalogChannel**.

UINT16 **GetTeamNumber ()**

Return the team number that the Driver Station is configured for.

**Dashboard & GetHighPriorityDashboardPacker ()**

**Dashboard & GetLowPriorityDashboardPacker ()**

**DashboardBase \* GetHighPriorityDashboardPackerInUse ()**

**DashboardBase \* GetLowPriorityDashboardPackerInUse ()**

void **SetHighPriorityDashboardPackerToUse  
(DashboardBase \*db)**

void **SetLowPriorityDashboardPackerToUse**

## **DriverStationEnhancedIO & *DriverStationIO*(\*db)**

void **IncrementUpdateNumber ()**

SEM\_ID **GetUserStatusDataSem ()**

void **InDisabled (bool entering)**

Only to be used to tell the Driver Station what code you claim to be executing for diagnostic purposes only.

void **InAutonomous (bool entering)**

Only to be used to tell the Driver Station what code you claim to be executing for diagnostic purposes only.

void **InOperatorControl (bool entering)**

Only to be used to tell the Driver Station what code you claim to be executing for diagnostic purposes only.

void **InTest (bool entering)**

Only to be used to tell the Driver Station what code you claim to be executing for diagnostic purposes only.

## **static DriverStation \* *GetInstance ()***

Return a pointer to the singleton **DriverStation**.

## Static Public Attributes

```
static const UINT32 kBatteryModuleNumber = 1  
static const UINT32 kBatteryChannel = 8  
static const UINT32 kJoystickPorts = 4  
static const UINT32 kJoystickAxes = 6
```

# Protected Member Functions

**DriverStation ()**

DriverStation contructor.

void **GetData ()**

Copy data from the DS task for the user.

void **SetData ()**

Copy status data from the DS task for the user.

## Detailed Description

Provide access to the network communication data to / from the Driver Station.

---

# Constructor & Destructor Documentation

## **DriverStation::DriverStation( )** [protected]

**DriverStation** contructor.

This is only called once the first time **GetInstance()** is called

# Member Function Documentation

## **DriverStation::Alliance** **DriverStation::GetAlliance( )**

Return the alliance that the driver station says it is on.

This could return kRed or kBlue

### **Returns:**

The Alliance enum

## **float DriverStation::GetAnalogIn ( UINT32 channel )**

Get an analog voltage from the Driver Station.

The analog values are returned as voltage 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.

## **float DriverStation::GetBatteryVoltage( )**

Read the battery voltage from the specified [AnalogChannel](#).

This accessor assumes that the battery voltage is being measured through the voltage divider on an analog breakout.

### **Returns:**

The battery voltage.

## **void DriverStation::GetData( ) [protected]**

Copy data from the DS task for the user.

If no new data exists, it will just be returned, otherwise the data will be copied from the DS polling loop.

## **bool DriverStation::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.

## **bool DriverStation::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.

## **UINT32 DriverStation::GetLocation ( )**

Return the driver station location on the field This could return 1, 2, or 3.

### **Returns:**

The location of the driver station

## **double DriverStation::GetMatchTime ( )**

Return the approximate match time The FMS does not currently send the official match time to the robots This returns the time since the enable signal sent from the Driver Station At the beginning of autonomous, the time is reset to 0.0 seconds At the beginning of teleop, the time is reset to +15.0 seconds If the robot is disabled, this returns 0.0 seconds Warning: This is not an official time (so it cannot be used to argue with referees)

**Returns:**

Match time in seconds since the beginning of autonomous

## **UINT32 DriverStation::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.

**Returns:**

The driver station packet number

## **float DriverStation::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.

## **short DriverStation::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.

## **UINT16 DriverStation::GetTeamNumber( )**

Return the team number that the Driver Station is configured for.

### **Returns:**

The team number

## **void DriverStation::InAutonomous( bool **entering** ) [inline]**

Only to be used to tell the Driver Station what code you claim to be executing for diagnostic purposes only.

### **Parameters:**

**entering** If true, starting autonomous code; if false, leaving autonomous code

## **void DriverStation::InDisabled( bool **entering** ) [inline]**

Only to be used to tell the Driver Station what code you claim to be executing for diagnostic purposes only.

### **Parameters:**

**entering** If true, starting disabled code; if false, leaving disabled code

## **void DriverStation::InOperatorControl( bool **entering** ) [inline]**

Only to be used to tell the Driver Station what code you claim to be executing for diagnostic purposes only.

### **Parameters:**

**entering** If true, starting teleop code; if false, leaving teleop code

## **void DriverStation::InTest( bool **entering** ) [inline]**

Only to be used to tell the Driver Station what code you claim to be executing for diagnostic purposes only.

### **Parameters:**

**entering** If true, starting test code; if false, leaving test code

## **bool DriverStation::IsFMSAttached ( )**

Is the driver station attached to a Field Management System? Note: This does not work with the Blue DS.

### **Returns:**

True if the robot is competing on a field being controlled by a Field Management System

## **bool DriverStation::IsNewControlData ( )**

Has a new control packet from the driver station arrived since the last time this function was called? Warning: If you call this function from more than one place at the same time, you will not get the intended behavior.

### **Returns:**

True if the control data has been updated since the last call.

## **void DriverStation::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.

## **void DriverStation::WaitForData ( )**

Wait until a new packet comes from the driver station This blocks on a semaphore, so the waiting is efficient.

This is a good way to delay processing until there is new driver station data to act on

---

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

- C:/WindRiver/workspace/WPILib/**DriverStation.h**
  - C:/WindRiver/workspace/WPILib/DriverStation.cpp
- 

Generated by  1.7.2

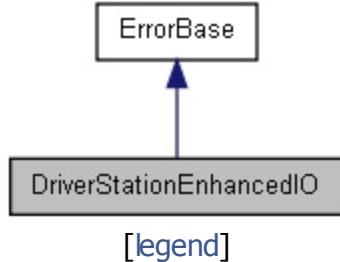


# DriverStationEnhancedIO Class Reference

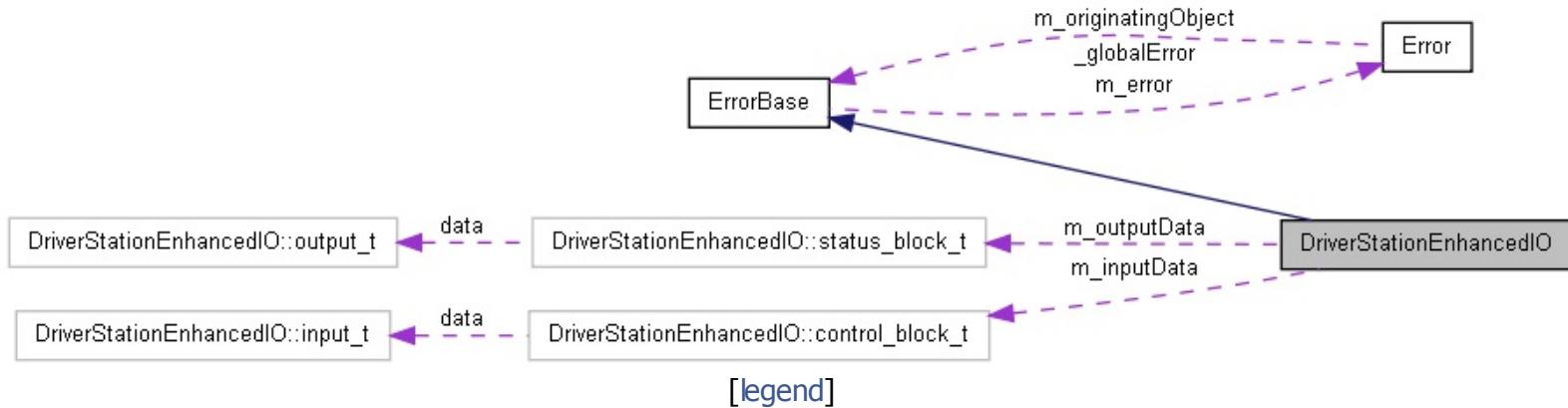
Interact with the more complete I/O available from the newest driver station. [More...](#)

```
#include <DriverStationEnhancedIO.h>
```

## Inheritance diagram for DriverStationEnhancedIO:



## Collaboration diagram for DriverStationEnhancedIO:



## List of all members.

## Classes

```
struct control_block_t
struct input_t
struct output_t
struct status_block_t
```

```
enum tDigitalConfig {
    kUnknown, kInputFloating, kInputPullUp, kInputPullDown,
    kOutput, kPWM, kAnalogComparator
}
enum tAccelChannel { kAccelX = 0, kAccelY = 1, kAccelZ = 2 }
tpWMPeriodChannels { kPWMChannels1and2,
enum kPWMChannels3and4 }
```

# Public Member Functions

- |        |  |   |
|--------|--|---|
| double | <b>GetAcceleration</b> (tAccelChannel channel)     | Query an accelerometer channel on the DS IO.                    |
| double | <b>GetAnalogIn</b> (UINT32 channel)                | Query an analog input channel on the DS IO.                     |
| double | <b>GetAnalogInRatio</b> (UINT32 channel)           | Query an analog input channel on the DS IO in ratiometric form. |
| double | <b>GetAnalogOut</b> (UINT32 channel)               | Query the voltage currently being output.                       |
| void   | <b>SetAnalogOut</b> (UINT32 channel, double value) | Set the analog output voltage.                                  |
| bool   | <b>GetButton</b> (UINT32 channel)                  | Get the state of a button on the IO board.                      |
| UINT8  | <b>GetButtons</b> ()                               | Get the state of all the button channels.                       |

	void	<b>SetLED</b> (UINT32 channel, bool value)	Set the state of an LED on the IO board.
	void	<b>SetLEDs</b> (UINT8 value)	Set the state of all 8 LEDs on the IO board.
	bool	<b>GetDigital</b> (UINT32 channel)	Get the current state of a DIO channel regardless of mode.
	UINT16	<b>GetDigitals</b> ()	Get the state of all 16 DIO lines regardless of mode.
	void	<b>SetDigitalOutput</b> (UINT32 channel, bool value)	Set the state of a DIO line that is configured for digital output.
tDigitalConfig		<b>GetDigitalConfig</b> (UINT32 channel)	Get the current configuration for a DIO line.
	void	<b>SetDigitalConfig</b> (UINT32 channel, tDigitalConfig config)	Override the DS's configuration of a DIO line.
	double	<b>GetPWMPPeriod</b> (tPWMPPeriodChannels channels)	Get the period of a <b>PWM</b> generator.
	void	<b>SetPWMPPeriod</b> (tPWMPPeriodChannels channels, double period)	Set the period of a <b>PWM</b> generator.
	bool	<b>GetFixedDigitalOutput</b> (UINT32 channel)	Get the state being output on a fixed digital output.
	void	<b>SetFixedDigitalOutput</b> (UINT32 channel, bool value)	Set the state to output on a Fixed High Current Digital Output line.
INT16		<b>GetEncoder</b> (UINT32 encoderNumber)	Get the position of a quadrature encoder.
	void	<b>ResetEncoder</b> (UINT32 encoderNumber)	Reset the position of an encoder to 0.
	bool	<b>GetEncoderIndexEnable</b> (UINT32 encoderNumber)	Get the current configuration of a quadrature encoder index channel.
	void	<b>SetEncoderIndexEnable</b> (UINT32 encoderNumber, bool enable)	Enable or disable the index channel of a quadrature encoder.
	double	<b>GetTouchSlider</b> ()	Get the value of the Capacitive Sense touch slider.
	double	<b>GetPWMOoutput</b> (UINT32 channel)	Get the percent duty-cycle that the <b>PWM</b> generator channel is configured to output.
	void	<b>SetPWMOoutput</b> (UINT32 channel, double value)	Set the percent duty-cycle to output on a <b>PWM</b> enabled DIO line.
UINT8		<b>GetFirmwareVersion</b> ()	Get the firmware version running on the IO board.

# **Friends**

---

class **DriverStation**

---

## Detailed Description

Interact with the more complete I/O available from the newest driver station.

Get a reference to an object of this type by calling GetEnhancedIO() on the **DriverStation** object.

---

# Member Function Documentation

## **double DriverStationEnhancedIO::GetAcceleration ( tAccelChannel **channel** )**

Query an accelerometer channel on the DS IO.

### **Parameters:**

**channel** The channel number to read.

### **Returns:**

The current acceleration on the channel in Gs.

## **double DriverStationEnhancedIO::GetAnalogIn ( UINT32 **channel** )**

Query an analog input channel on the DS IO.

### **Parameters:**

**channel** The channel number to read. [1,8]

### **Returns:**

The analog input voltage for the channel.

## **double DriverStationEnhancedIO::GetAnalogInRatio ( UINT32 **channel** )**

Query an analog input channel on the DS IO in ratiometric form.

### **Parameters:**

**channel** The channel number to read. [1,8]

### **Returns:**

The analog input percentage for the channel.

## **double DriverStationEnhancedIO::GetAnalogOut ( UINT32 **channel** )**

Query the voltage currently being output.

AO1 is pin 11 on the top connector (P2). AO2 is pin 12 on the top connector (P2).

### **Parameters:**

**channel** The analog output channel on the DS IO. [1,2]

**Returns:**

The voltage being output on the channel.

**bool DriverStationEnhancedIO::GetButton ( UINT32 channel )**

Get the state of a button on the IO board.

Button1 is the physical button "S1". Button2 is pin 4 on the top connector (P2). Button3 is pin 6 on the top connector (P2). Button4 is pin 8 on the top connector (P2). Button5 is pin 10 on the top connector (P2). Button6 is pin 7 on the top connector (P2).

Button2 through Button6 are Capacitive Sense buttons.

**Parameters:**

**channel** The button channel to read. [1,6]

**Returns:**

The state of the selected button.

**UINT8 DriverStationEnhancedIO::GetButtons ( )**

Get the state of all the button channels.

**Returns:**

The state of the 6 button channels in the 6 lsb of the returned byte.

**bool DriverStationEnhancedIO::GetDigital ( UINT32 channel )**

Get the current state of a DIO channel regardless of mode.

**Parameters:**

**channel** The DIO channel to read. [1,16]

**Returns:**

The state of the selected digital line.

**DriverStationEnhancedIO::tDigitalConfig DriverStationEnhancedIO::GetDigitalConfig ( )**

Get the current configuration for a DIO line.

This has the side effect of forcing the Driver Station to switch to Enhanced mode if it's not when called. If Enhanced mode is not enabled when this is called, it will return kUnknown.

**Parameters:**

**channel** The DIO channel config to get. [1,16]

**Returns:**

The configured mode for the DIO line.

## **UINT16 DriverStationEnhancedIO::GetDigitals( )**

Get the state of all 16 DIO lines regardless of mode.

**Returns:**

The state of all DIO lines. DIO1 is lsb and DIO16 is msb.

## **INT16 DriverStationEnhancedIO::GetEncoder( UINT32 encoderNumber )**

Get the position of a quadrature encoder.

There are two signed 16-bit 4X quadrature decoders on the IO board. These decoders are always monitoring the state of the lines assigned to them, but these lines do not have to be used for encoders.

Encoder1 uses DIO4 for "A", DIO6 for "B", and DIO8 for "Index". Encoder2 uses DIO5 for "A", DIO7 for "B", and DIO9 for "Index".

The index functionality can be enabled or disabled using  
[\*\*SetEncoderIndexEnable\(\)\*\*](#).

**Parameters:**

**encoderNumber** The quadrature encoder to access. [1,2]

**Returns:**

The current position of the quadrature encoder.

## **bool DriverStationEnhancedIO::GetEncoderIndexEnable( UINT32 encoderNu**

Get the current configuration of a quadrature encoder index channel.

This has the side effect of forcing the Driver Station to switch to Enhanced mode if it's not when called. If Enhanced mode is not enabled when this is called, it will return false.

**Parameters:**

**encoderNumber** The quadrature encoder. [1,2]

**Returns:**

Is the index channel of the encoder enabled.

## **UINT8 DriverStationEnhancedIO::GetFirmwareVersion( )**

Get the firmware version running on the IO board.

This also has the side effect of forcing the driver station to switch to Enhanced mode if it is not. If you plan to switch between Driver Stations with unknown IO configurations, you can call this until it returns a non-0 version to ensure that this API is accessible before proceeding.

**Returns:**

The version of the firmware running on the IO board. 0 if the board is not attached or not in Enhanced mode.

## **bool DriverStationEnhancedIO::GetFixedDigitalOutput( UINT32 channel )**

Get the state being output on a fixed digital output.

**Parameters:**

**channel** The FixedDO line to get. [1,2]

**Returns:**

The state of the FixedDO line.

## **double DriverStationEnhancedIO::GetPWMOuput( UINT32 channel )**

Get the percent duty-cycle that the **PWM** generator channel is configured to output.

**Parameters:**

**channel** The DIO line's **PWM** generator to get the duty-cycle from. [1,4]

**Returns:**

The percent duty-cycle being output (if the DIO line is configured for **PWM**). [0.0,1.0]

## **double DriverStationEnhancedIO::GetPWMPPeriod ( tPWMPPeriodChannels channels )**

Get the period of a **PWM** generator.

This has the side effect of forcing the Driver Station to switch to Enhanced mode if it's not when called. If Enhanced mode is not enabled when this is called, it will return 0.

**Parameters:**

**channels** Select the generator by specifying the two channels to which it is connected.

**Returns:**

The period of the **PWM** generator in seconds.

## **double DriverStationEnhancedIO::GetTouchSlider ( )**

Get the value of the Capacitive Sense touch slider.

**Returns:**

Value between 0.0 (toward center of board) and 1.0 (toward edge of board). -1.0 means no touch detected.

## **void DriverStationEnhancedIO::ResetEncoder ( UINT32 encoderNumber )**

Reset the position of an encoder to 0.

This simply stores an offset locally. It does not reset the hardware counter on the IO board. If you use this method with Index enabled, you may get unexpected results.

**Parameters:**

**encoderNumber** The quadrature encoder to reset. [1,2]

## **void DriverStationEnhancedIO::SetAnalogOut ( UINT32 channel,**

**double value**

)

Set the analog output voltage.

AO1 is pin 11 on the top connector (P2). AO2 is pin 12 on the top connector (P2). AO1 is the reference voltage for the 2 analog comparators on DIO15 and DIO16.

The output range is 0V to 4V, however due to the supply voltage don't expect more than about 3V. Current supply capability is only 100uA.

### Parameters:

**channel** The analog output channel on the DS IO. [1,2]  
**value** The voltage to output on the channel.

**void DriverStationEnhancedIO::SetDigitalConfig ( UINT32 channel, tDigitalConfig config )**

Override the DS's configuration of a DIO line.

If configured to kInputFloating, the selected DIO line will be tri-stated with no internal pull resistor.

If configured to kInputPullUp, the selected DIO line will be tri-stated with a 5k-Ohm internal pull-up resistor enabled.

If configured to kInputPullDown, the selected DIO line will be tri-stated with a 5k-Ohm internal pull-down resistor enabled.

If configured to kOutput, the selected DIO line will actively drive to 0V or Vddio (specified by J1 and J4). DIO1 through DIO12, DIO15, and DIO16 can source 4mA and can sink 8mA. DIO12 and DIO13 can source 4mA and can sink 25mA.

In addition to the common configurations, DIO1 through DIO4 can be configured to kPWM to enable **PWM** output.

In addition to the common configurations, DIO15 and DIO16 can be configured to kAnalogComparator to enable analog comparators on those 2 DIO lines. When enabled, the lines are tri-stated and will accept analog voltages between 0V and 3.3V. If the input voltage is greater than the voltage output by AO1, the DIO will read as true, if less then false.

## Parameters:

**channel** The DIO line to configure. [1,16]

**config** The mode to put the DIO line in.

```
void DriverStationEnhancedIO::SetDigitalOutput( UINT32 channel,  
                                              bool      value  
                                              )
```

Set the state of a DIO line that is configured for digital output.

## Parameters:

**channel** The DIO channel to set. [1,16]

**value** The state to set the selected channel to.

```
void DriverStationEnhancedIO::SetEncoderIndexEnable( UINT32 encoderNu  
                                              bool      enable  
                                              )
```

Enable or disable the index channel of a quadrature encoder.

The quadrature decoders on the IO board support an active-low index input.

Encoder1 uses DIO8 for "Index". Encoder2 uses DIO9 for "Index".

When enabled, the decoder's counter will be reset to 0 when A, B, and Index are all low.

## Parameters:

**encoderNumber** The quadrature encoder. [1,2]

**enable** If true, reset the encoder in an index condition.

```
void DriverStationEnhancedIO::SetFixedDigitalOutput( UINT32 channel,  
                                              bool      value  
                                              )
```

Set the state to output on a Fixed High Current Digital Output line.

FixedDO1 is pin 5 on the top connector (P2). FixedDO2 is pin 3 on the top connector (P2).

The FixedDO lines always output 0V and 3.3V regardless of J1 and J4. They can source 4mA and can sink 25mA. Because of this, they are expected to be used in an active low configuration, such as connecting to the cathode of a bright LED. Because they are expected to be active low, they default to true.

### Parameters:

**channel** The FixedDO channel to set.

**value** The state to set the FixedDO.

```
void DriverStationEnhancedIO::SetLED ( UINT32 channel,  
                                      bool      value  
                                    )
```

Set the state of an LED on the IO board.

### Parameters:

**channel** The LED channel to set. [1,8]

**value** True to turn the LED on.

```
void DriverStationEnhancedIO::SetLEDs ( UINT8 value )
```

Set the state of all 8 LEDs on the IO board.

### Parameters:

**value** The state of each LED. LED1 is lsb and LED8 is msb.

```
void DriverStationEnhancedIO::SetPWMOuput ( UINT32 channel,  
                                            double   value  
                                          )
```

Set the percent duty-cycle to output on a **PWM** enabled DIO line.

DIO1 through DIO4 have the ability to output a **PWM** signal. The period of the signal can be configured in pairs using **SetPWMPeriod()**.

### Parameters:

**channel** The DIO line's **PWM** generator to set. [1,4]

**value** The percent duty-cycle to output from the **PWM** generator. [0.0,1.0]

```
void DriverStationEnhancedIO::SetPWMPulseWidthModulationPeriod(tPWMPulseWidthModulationChannels channels, double period)
```

Set the period of a **PWM** generator.

There are 2 **PWM** generators on the IO board. One can generate **PWM** signals on DIO1 and DIO2, the other on DIO3 and DIO4. Each generator has one counter and two compare registers. As such, each pair of **PWM** outputs share the output period but have independent duty cycles.

#### Parameters:

**channels** Select the generator by specifying the two channels to which it is connected.

**period** The period of the **PWM** generator in seconds. [0.0,0.002731]

---

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

- C:/WindRiver/workspace/WPIlib/**DriverStationEnhancedIO.h**
- C:/WindRiver/workspace/WPIlib/DriverStationEnhancedIO.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

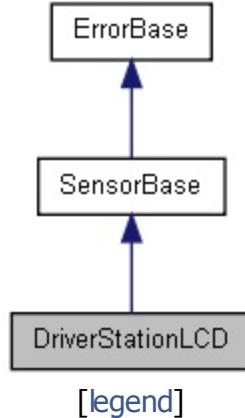
[Public Types](#) | [Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Static Public Attributes](#) |  
[Protected Member Functions](#)

# DriverStationLCD Class Reference

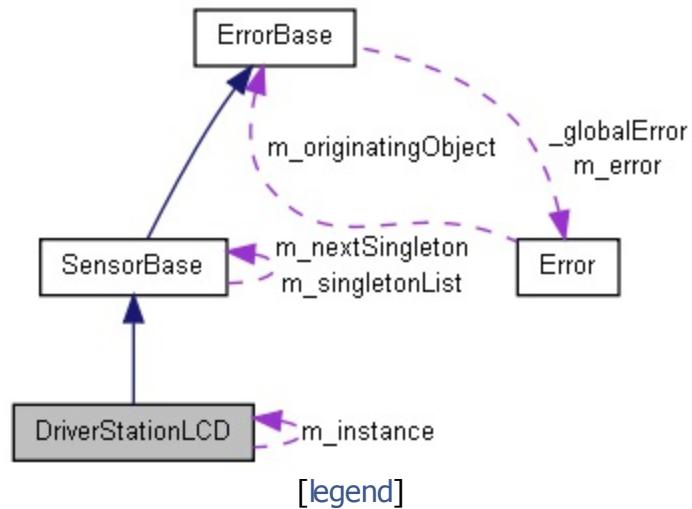
Provide access to "LCD" on the Driver Station. [More...](#)

```
#include <DriverStationLCD.h>
```

Inheritance diagram for DriverStationLCD:



Collaboration diagram for DriverStationLCD:



List of all members.

# Public Types

```
enum Line {
    kMain_Line6 = 0, kUser_Line1 = 0, kUser_Line2 =
    1, kUser_Line3 = 2,
    kUser_Line4 = 3, kUser_Line5 = 4, kUser_Line6 =
    5
}
```

void **UpdateLCD ()**

Send the text data to the Driver Station.

void **Printf (Line line, INT32 startingColumn, const char \*writeFmt,...)**

Print formatted text to the Driver Station LCD text bufer.

void **VPrintf (Line line, INT32 startingColumn, const char \*writeFmt, va\_list args)**

void **PrintfLine (Line line, const char \*writeFmt,...)**

Print formatted text to the Driver Station LCD text bufer.

void **VPrintfLine (Line line, const char \*writeFmt, va\_list args)**

void **Clear ()**

Clear all lines on the LCD.

# Static Public Member Functions

---

static **DriverStationLCD** \* **GetInstance** ()

Return a pointer to the singleton **DriverStationLCD**.

## Static Public Attributes

```
static const UINT32 kSyncTimeout_ms = 20  
static const UINT16 kFullDisplayTextCommand = 0x9FF  
static const INT32 kLineLength = 21  
static const INT32 kNumLines = 6
```

**DriverStationLCD ()**  
**DriverStationLCD** contructor.

## Detailed Description

Provide access to "LCD" on the Driver Station.

This is the Messages box on the DS Operation tab.

Buffer the printed data locally and then send it when UpdateLCD is called.

---

# Constructor & Destructor Documentation

## **DriverStationLCD::DriverStationLCD( )** [protected]

**DriverStationLCD** contructor.

This is only called once the first time **GetInstance()** is called

# Member Function Documentation

```
void DriverStationLCD::Printf( Line           line,
                               INT32          startingColumn,
                               const char *   writeFmt,
                               ... )
)
```

Print formatted text to the Driver Station LCD text bufer.

Use [UpdateLCD\(\)](#) periodically to actually send the text to the Driver Station.

## Parameters:

**line** The line on the LCD to print to.  
**startingColumn** The column to start printing to. This is a 1-based number.  
**writeFmt** The printf format string describing how to print.

```
void DriverStationLCD::PrintfLine( Line           line,
                                   const char *   writeFmt,
                                   ... )
)
```

Print formatted text to the Driver Station LCD text bufer.

This function pads the line with empty spaces.

Use [UpdateLCD\(\)](#) periodically to actually send the text to the Driver Station.

## Parameters:

**line** The line on the LCD to print to.  
**writeFmt** The printf format string describing how to print.

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

- C:/WindRiver/workspace/WPIlib/[DriverStationLCD.h](#)
- C:/WindRiver/workspace/WPIlib/DriverStationLCD.cpp

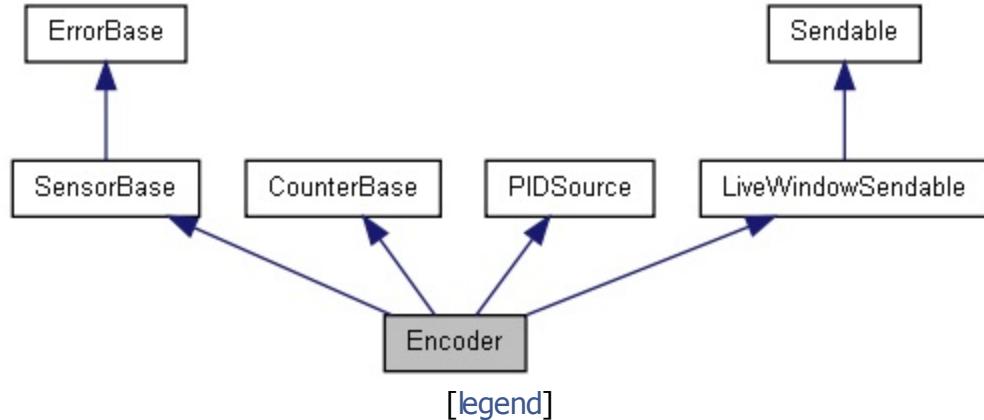


# Encoder Class Reference

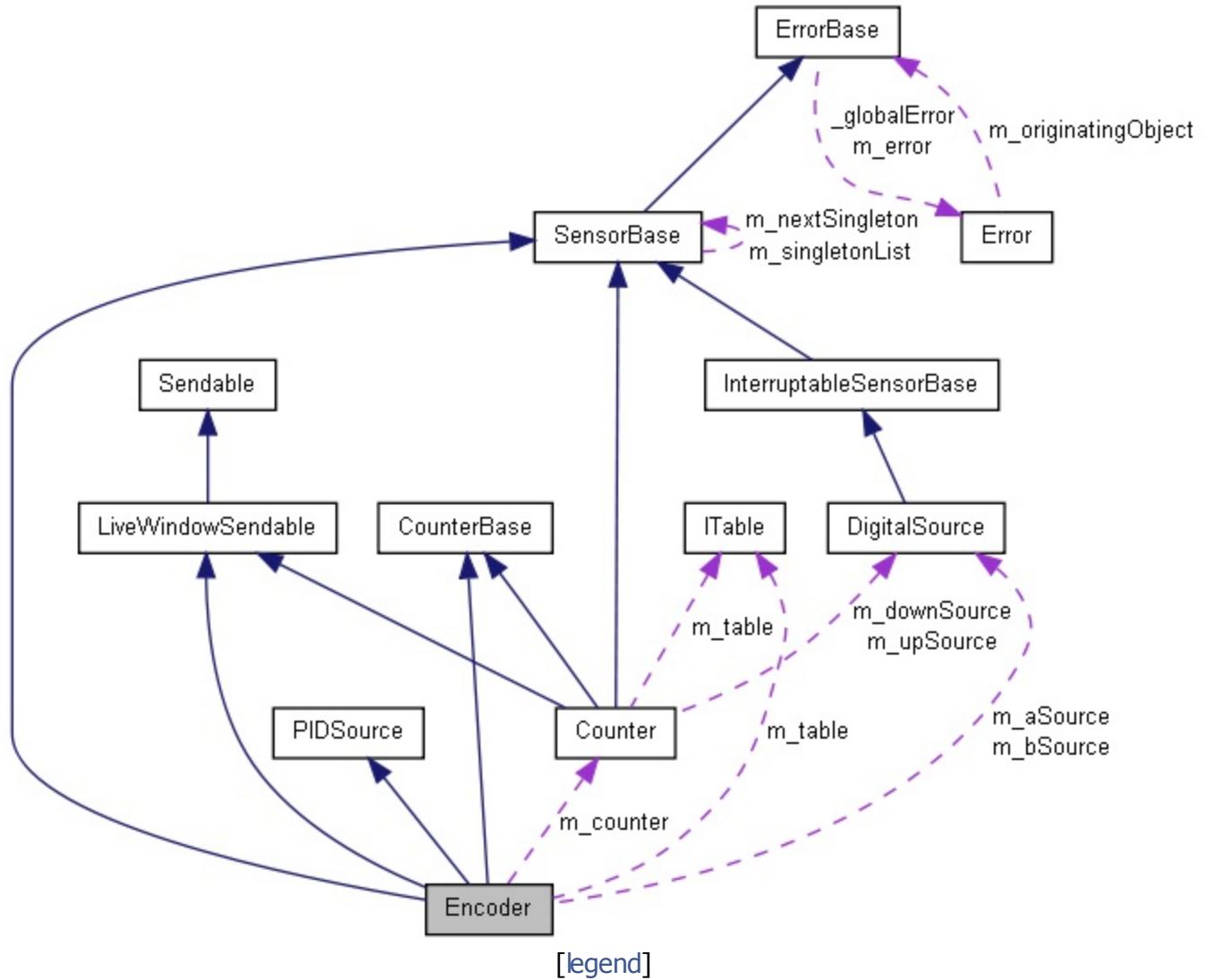
Class to read quad encoders. More...

```
#include <Encoder.h>
```

Inheritance diagram for Encoder:



Collaboration diagram for Encoder:



List of all members.

## Public Types

```
enum PIDSourceParameter { kDistance, kRate }
```

# Public Member Functions

**Encoder** (UINT32 aChannel, UINT32 bChannel, bool reverseDirection=false, EncodingType encodingType=k4X)  
**Encoder** constructor.

**Encoder** (UINT8 aModuleNumber, UINT32 aChannel, UINT8 bModuleNumber, UINT32 \_bChannel, bool reverseDirection=false, EncodingType encodingType=k4X)  
**Encoder** constructor.

**Encoder** (**DigitalSource** \*aSource, **DigitalSource** \*bSource, bool reverseDirection=false, EncodingType encodingType=k4X)  
**Encoder** constructor.

**Encoder** (**DigitalSource** &aSource, **DigitalSource** &bSource, bool reverseDirection=false, EncodingType encodingType=k4X)  
**Encoder** constructor.

virtual **~Encoder** ()  
Free the resources for an **Encoder**.

void **Start** ()  
Start the **Encoder**.

INT32 **Get** ()  
Gets the current count.

INT32 **GetRaw** ()  
Gets the raw value from the encoder.

void **Reset** ()  
Reset the **Encoder** distance to zero.

void **Stop** ()  
Stops counting pulses on the **Encoder** device.

double **GetPeriod** ()  
Returns the period of the most recent pulse.

void **SetMaxPeriod** (double maxPeriod)  
Sets the maximum period for stopped detection.

bool **GetStopped** ()  
Determine if the encoder is stopped.

bool **GetDirection** ()  
The last direction the encoder value changed.

double **GetDistance** ()  
Get the distance the robot has driven since the last reset.

double **GetRate** ()

void	<b>GetMinRate ()</b>	Get the current rate of the encoder.
	<b>SetMinRate (double minRate)</b>	Set the minimum rate of the device before the hardware reports it stopped.
void	<b>SetDistancePerPulse (double distancePerPulse)</b>	Set the distance per pulse for this encoder.
void	<b>SetReverseDirection (bool reverseDirection)</b>	Set the direction sensing for this encoder.
void	<b>SetPIDSourceParameter (PIDSourceParameter pidSource)</b>	Set which parameter of the encoder you are using as a process control variable.
double	<b>PIDGet ()</b>	Implement the <b>PIDSource</b> interface.
void	<b>UpdateTable ()</b>	Update the table for this sendable object with the latest values.
void	<b>StartLiveWindowMode ()</b>	Start having this sendable object automatically respond to value changes reflect the value on the table.
void	<b>StopLiveWindowMode ()</b>	Stop having this sendable object automatically respond to value changes.
std::string	<b>GetSmartDashboardType ()</b>	
void	<b>InitTable (ITable *subTable)</b>	Initializes a table for this sendable object.
<b>ITable *</b>	<b>GetTable ()</b>	

## Detailed Description

Class to read quad encoders.

Quadrature encoders are devices that count shaft rotation and can sense direction. The output of the QuadEncoder class is an integer that can count either up or down, and can go negative for reverse direction counting. When creating QuadEncoders, a direction is supplied that changes the sense of the output to make code more readable if the encoder is mounted such that forward movement generates negative values. Quadrature encoders have two digital outputs, an A Channel and a B Channel that are out of phase with each other to allow the FPGA to do direction sensing.

---

# Constructor & Destructor Documentation

```
Encoder::Encoder ( UINT32           aChannel,  
                  UINT32           bChannel,  
                  bool            reverseDirection = false,  
                  EncodingType encodingType = k4x )
```

**Encoder** constructor.

Construct a **Encoder** given a and b channels assuming the default module.

## Parameters:

<b>aChannel</b>	The a channel digital input channel.
<b>bChannel</b>	The b channel digital input channel.
<b>reverseDirection</b>	represents the orientation of the encoder and inverts the output values if necessary so forward represents positive values.
<b>encodingType</b>	either k1X, k2X, or k4X to indicate 1X, 2X or 4X decoding. If 4X is selected, then an encoder FPGA object is used and the returned counts will be 4x the encoder spec'd value since all rising and falling edges are counted. If 1X or 2X are selected then a counter object will be used and the returned value will either exactly match the spec'd count or be double (2x) the spec'd count.

```
Encoder::Encoder ( UINT8          aModuleNumber,  
                  UINT32        aChannel,  
                  UINT8          bModuleNumber,  
                  UINT32        bChannel,  
                  bool          reverseDirection = false,  
                  EncodingType encodingType = k4x )
```

**Encoder** constructor.

Construct a **Encoder** given a and b modules and channels fully specified.

## Parameters:

<b>aModuleNumber</b>	The a channel digital input module.
<b>aChannel</b>	The a channel digital input channel.
<b>bModuleNumber</b>	The b channel digital input module.
<b>bChannel</b>	The b channel digital input channel.
<b>reverseDirection</b>	represents the orientation of the encoder and inverts the output values if necessary so forward represents positive values.
<b>encodingType</b>	either k1X, k2X, or k4X to indicate 1X, 2X or 4X decoding. If 4X is selected, then an encoder FPGA object is used and the returned counts will be 4x the encoder spec'd value since all rising and falling edges are counted. If 1X or 2X are selected then a counter object will be used and the returned value will either exactly match the spec'd count or be double (2x) the spec'd count.

```
Encoder::Encoder ( DigitalSource * aSource,
                    DigitalSource * bSource,
                    bool            reverseDirection = false,
                    EncodingType    encodingType = k4x
)
```

**Encoder** constructor.

Construct a **Encoder** given a and b channels as digital inputs. This is used in the case where the digital inputs are shared. The **Encoder** class will not allocate the digital inputs and assume that they already are counted.

#### Parameters:

<b>aSource</b>	The source that should be used for the a channel.
<b>bSource</b>	the source that should be used for the b channel.
<b>reverseDirection</b>	represents the orientation of the encoder and inverts the output values if necessary so forward represents positive values.
<b>encodingType</b>	either k1X, k2X, or k4X to indicate 1X, 2X or 4X decoding. If 4X is selected, then an encoder FPGA object is used and the returned counts will be 4x the encoder spec'd value since all rising and falling edges are counted. If 1X or 2X are selected then a counter object will be used and the returned value will either exactly match the spec'd count or be double (2x) the spec'd count.

```
Encoder::Encoder( DigitalSource & aSource,
                  DigitalSource & bSource,
                  bool                reverseDirection = false,
                  EncodingType        encodingType = k4x
                )
```

## Encoder constructor.

Construct a **Encoder** given a and b channels as digital inputs. This is used in the case where the digital inputs are shared. The **Encoder** class will not allocate the digital inputs and assume that they already are counted.

### Parameters:

<b>aSource</b>	The source that should be used for the a channel.
<b>bSource</b>	the source that should be used for the b channel.
<b>reverseDirection</b>	represents the orientation of the encoder and inverts the output values if necessary so forward represents positive values.
<b>encodingType</b>	either k1X, k2X, or k4X to indicate 1X, 2X or 4X decoding. If 4X is selected, then an encoder FPGA object is used and the returned counts will be 4x the encoder spec'd value since all rising and falling edges are counted. If 1X or 2X are selected then a counter object will be used and the returned value will either exactly match the spec'd count or be double (2x) the spec'd count.

## Encoder::~Encoder( ) [virtual]

Free the resources for an **Encoder**.

Frees the FPGA resources associated with an **Encoder**.

# Member Function Documentation

## **INT32 Encoder::Get( ) [virtual]**

Gets the current count.

Returns the current count on the [Encoder](#). This method compensates for the decoding type.

### **Returns:**

Current count from the [Encoder](#) adjusted for the 1x, 2x, or 4x scale factor.

Implements [CounterBase](#).

## **bool Encoder::GetDirection( ) [virtual]**

The last direction the encoder value changed.

### **Returns:**

The last direction the encoder value changed.

Implements [CounterBase](#).

## **double Encoder::GetDistance( )**

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 [SetDistancePerPulse\(\)](#).

## **double Encoder::GetPeriod( ) [virtual]**

Returns the period of the most recent pulse.

Returns the period of the most recent [Encoder](#) pulse in seconds. This method compensates for the decoding type.

### **Deprecated:**

Use [GetRate\(\)](#) in favor of this method. This returns unscaled periods and [GetRate\(\)](#) scales using value from [SetDistancePerPulse\(\)](#).

**Returns:**

Period in seconds of the most recent pulse.

Implements [CounterBase](#).

**double Encoder::GetRate( )**

Get the current rate of the encoder.

Units are distance per second as scaled by the value from [SetDistancePerPulse\(\)](#).

**Returns:**

The current rate of the encoder.

**INT32 Encoder::GetRaw( )**

Gets the raw value from the encoder.

The raw value is the actual count unscaled by the 1x, 2x, or 4x scale factor.

**Returns:**

Current raw count from the encoder

**std::string Encoder::GetSmartDashboardType( ) [virtual]****Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements [Sendable](#).

**bool Encoder::GetStopped( ) [virtual]**

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.

**Returns:**

True if the encoder is considered stopped.

Implements **CounterBase**.

### **ITable \* Encoder::GetTable( ) [virtual]**

**Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

### **void Encoder::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

**Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

### **double Encoder::PIDGet( ) [virtual]**

Implement the **PIDSource** interface.

**Returns:**

The current value of the selected source parameter.

Implements **PIDSource**.

### **void Encoder::Reset( ) [virtual]**

Reset the **Encoder** distance to zero.

Resets the current count to zero on the encoder.

Implements **CounterBase**.

### **void Encoder::SetDistancePerPulse( 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.

### **void Encoder::SetMaxPeriod ( double maxPeriod ) [virtual]**

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 **SetMinRate()** in favor of this method. This takes unscaled periods and **SetMinRate()** scales using value from **SetDistancePerPulse()**.

#### Parameters:

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

Implements **CounterBase**.

### **void Encoder::SetMinRate ( 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 **SetDistancePerPulse()**.

## **void Encoder::SetPIDSourceParameter ( PIDSourceParameter pidSource )**

Set which parameter of the encoder you are using as a process control variable.

### **Parameters:**

**pidSource** An enum to select the parameter.

## **void Encoder::SetReverseDirection ( 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:**

**reverseDirection** true if the encoder direction should be reversed

## **void Encoder::Start( ) [virtual]**

Start the **Encoder**.

Starts counting pulses on the **Encoder** device.

Implements **CounterBase**.

## **void Encoder::Stop( ) [virtual]**

Stops counting pulses on the **Encoder** device.

The value is not changed.

Implements **CounterBase**.

---

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

- C:/WindRiver/workspace/WPIlib/**Encoder.h**
- C:/WindRiver/workspace/WPIlib/Encoder.cpp



# EntryCache Class Reference

---

List of all members.

# Public Member Functions

**EntryCache** (std::string &path)

**NetworkTableEntry** \* **Get** (std::string &key)

---

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

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTable.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTable.cpp

---

Generated by  1.7.2



# **EntryValue Union Reference**

---

List of all members.

## Public Attributes

void *	<b>ptr</b>
bool	<b>b</b>
double	<b>f</b>

---

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

- C:/WindRiver/workspace/WPILib/tables/**ITable.h**

---

Generated by [doxygen](#) 1.7.2

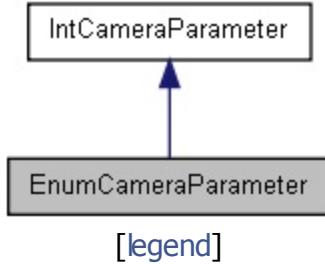


# EnumCameraParameter Class Reference

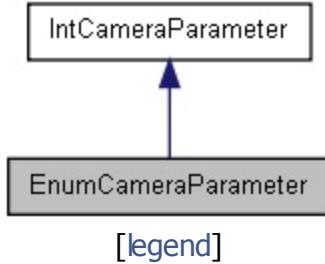
Enumerated camera parameter. [More...](#)

```
#include <EnumCameraParameter.h>
```

Inheritance diagram for EnumCameraParameter:



Collaboration diagram for EnumCameraParameter:



[List of all members.](#)

## Public Member Functions

**EnumCameraParameter** (const char \*setString, const char \*getString,  
bool requiresRestart, const char \*const \*choices, int numChoices)  
Constructor for an enumeration camera parameter.

virtual bool **CheckChanged** (bool &changed, char \*param)  
Check if a parameter has changed and update.

virtual void **GetParamFromString** (const char \*string, int stringLength)  
Extract the parameter value from a string.

## Detailed Description

Enumerated camera parameter.

This class represents a camera parameter that takes an enumerated type for a value.

---

# Constructor & Destructor Documentation

```
EnumCameraParameter::EnumCameraParameter( const char *  
                                         const char *  
                                         bool  
                                         const char *const *  
                                         int  
                                         )
```

Constructor for an enumeration camera parameter.

Enumeration camera parameters have lists of value choices and strings that go with them. There are also C++ enumerations to go along with them.

## Parameters:

- setString** The string for an HTTP request to set the value.
- getString** The string for an HTTP request to get the value.
- choices** An array of strings of the parameter choices set in the http strings.
- numChoices** The number of choices in the enumeration set.

# Member Function Documentation

```
bool EnumCameraParameter::CheckChanged ( bool & changed,  
                                         char * param  
                                         ) [virtual]
```

Check if a parameter has changed and update.

Check if a parameter has changed and send the update string if it has changed. This is called from the loop in the parameter task loop.

## Returns:

true if the camera needs to restart

Reimplemented from [IntCameraParameter](#).

```
void EnumCameraParameter::GetParamFromString ( const char * string,  
                                              int stringLength  
                                              ) [virtual]
```

Extract the parameter value from a string.

Extract the parameter value from the camera status message.

## Parameters:

**string** The string returned from the camera.

**length** The length of the string from the camera.

Reimplemented from [IntCameraParameter](#).

---

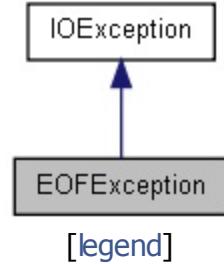
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/Vision/[EnumCameraParameter.h](#)
- C:/WindRiver/workspace/WPILib/Vision/EnumCameraParameter.cpp

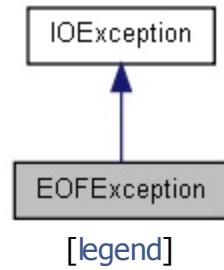


# EOFException Class Reference

Inheritance diagram for EOFException:



Collaboration diagram for EOFException:



List of all members.

# Public Member Functions

virtual bool **isEOF ()**

---

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

- C:/WindRiver/workspace/WPIlib/networktables2/util/**EOFException.h**
- C:/WindRiver/workspace/WPIlib/networktables2/util/EOFException.cpp

---

Generated by  1.7.2

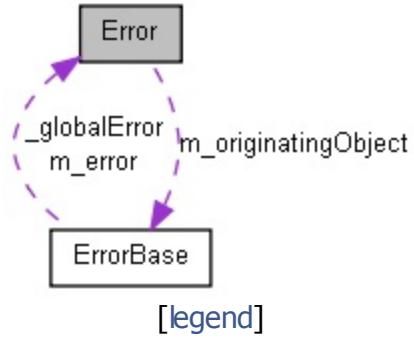
[Public Types](#) | [Public Member Functions](#) |  
[Static Public Member Functions](#)

# Error Class Reference

**Error** object represents a library error. [More...](#)

```
#include <Error.h>
```

Collaboration diagram for Error:



[List of all members.](#)

# Public Types

typedef tRioStatusCode **Code**

void	<b>Clone</b> ( <a href="#">Error</a> &error)
Code	<b>GetCode</b> () const
const char *	<b>GetMessage</b> () const
const char *	<b>GetFilename</b> () const
const char *	<b>GetFunction</b> () const
UINT32	<b>GetLineNumber</b> () const
const <a href="#">ErrorBase</a> *	<b>GetOriginatingObject</b> () const
double	<b>GetTime</b> () const
void	<b>Clear</b> ()
	<b>Set</b> (Code code, const char *contextMessage, const char *
	filename, const char *function, UINT32 lineNumber, const
	<a href="#">ErrorBase</a> *originatingObject)

# Static Public Member Functions

---

static void **EnableStackTrace** (bool enable)

static void **EnableSuspendOnError** (bool enable)

---

# Detailed Description

**Error** object represents a library error.

---

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

- C:/WindRiver/workspace/WPILib/**Error.h**
- C:/WindRiver/workspace/WPILib/Error.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

Public Member Functions |  
Static Public Member Functions |  
Protected Member Functions |  
Protected Attributes |  
Static Protected Attributes

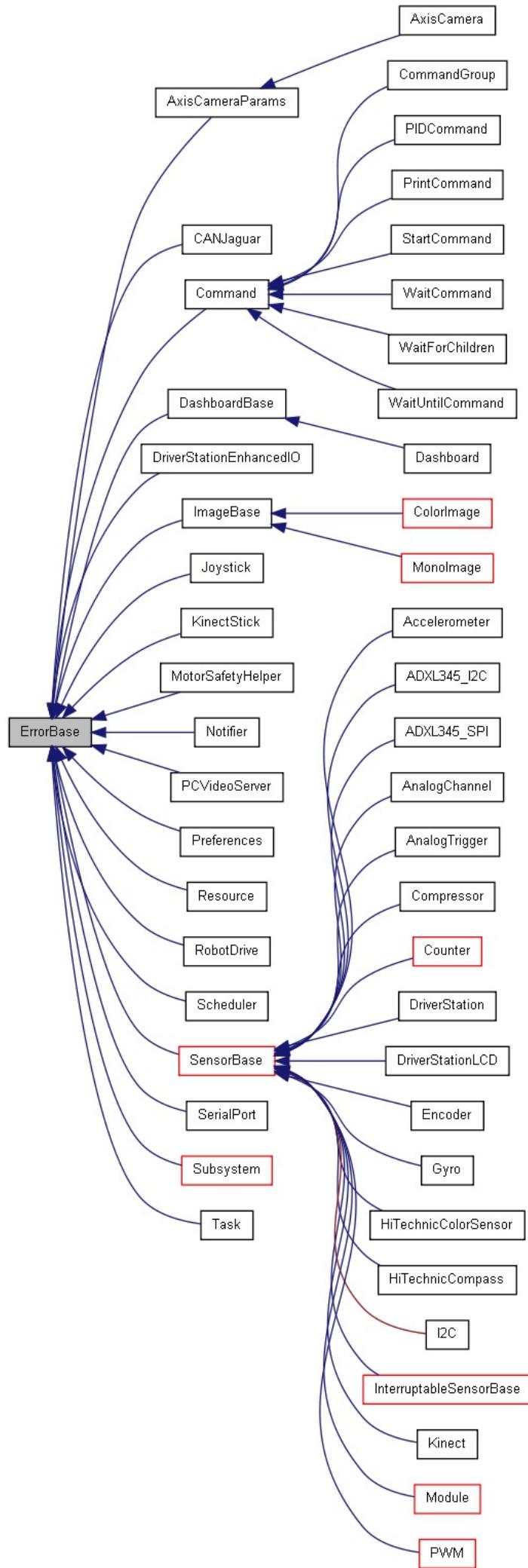
# ErrorBase Class Reference

---

Base class for most objects. [More...](#)

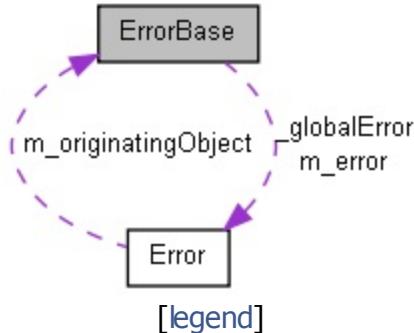
```
#include <ErrorBase.h>
```

Inheritance diagram for ErrorBase:



[legend]

## Collaboration diagram for ErrorBase:



List of all members.

**virtual Error & GetError ()**

Retrieve the current error.

**virtual const Error & GetError () const**

**virtual void SetErrnoError (const char \*contextMessage, const char \*filename, const char \*function, UINT32 lineNumber) const**  
Set error information associated with a C library call that set an error to the "errno" global variable.

**virtual void SetImaqError (int success, const char \*contextMessage, const char \*filename, const char \*function, UINT32 lineNumber) const**  
Set the current error information associated from the nivision Imaq API.

**virtual void SetError (Error::Code code, const char \*contextMessage, const char \*filename, const char \*function, UINT32 lineNumber) const**  
Set the current error information associated with this sensor.

**virtual void SetWPIError (const char \*errorMessage, const char \*contextMessage, const char \*filename, const char \*function, UINT32 lineNumber) const**  
Set the current error information associated with this sensor.

**virtual void CloneError (ErrorBase \*rhs) const**

**virtual void ClearError () const**

Clear the current error information associated with this sensor.

**virtual bool StatusIsFatal () const**

Check if the current error code represents a fatal error.

# Static Public Member Functions

**SetGlobalError** (Error::Code code, const char \*contextMessage, const char \*filename, const char \*function, UINT32 lineNumber)

**SetGlobalWPIError** (const char \*errorMessage, const char \*contextMessage, const char \*filename, const char \*function, UINT32 lineNumber)

static **Error** & **GetGlobalError** ()

Retrieve the current global error.

**ErrorBase** ()

Initialize the instance status to 0 for now.

## Protected Attributes

### Error **m\_error**

```
static SEM_ID _globalErrorMutex = semMCreate(SEM_Q_PRIORITY |  
SEM_DELETE_SAFE | SEM_INVERSION_SAFE)  
static Error _globalError
```

## Detailed Description

Base class for most objects.

**ErrorBase** is the base class for most objects since it holds the generated error for that object. In addition, there is a single instance of a global error object

---

# Member Function Documentation

## Error & ErrorBase::GetError ( ) [virtual]

Retrieve the current error.

Get the current error information associated with this sensor.

## void ErrorBase::SetErrnoError ( const char \* contextMessage,                                   const char \* filename,                                   const char \* function,                                   UINT32       lineNumber                                   )                          const [virtual]

Set error information associated with a C library call that set an error to the "errno" global variable.

### Parameters:

<b>contextMessage</b>	A custom message from the code that set the error.
<b>filename</b>	Filename of the error source
<b>function</b>	Function of the error source
<b>lineNumber</b>	Line number of the error source

## void ErrorBase::SetError ( Error::Code code,                                   const char \* contextMessage,                                   const char \* filename,                                   const char \* function,                                   UINT32       lineNumber                                   )                          const [virtual]

Set the current error information associated with this sensor.

### Parameters:

<b>code</b>	The error code
<b>contextMessage</b>	A custom message from the code that set the error.
<b>filename</b>	Filename of the error source
<b>function</b>	Function of the error source
<b>lineNumber</b>	Line number of the error source

```
void ErrorBase::SetImaqError ( int success,
                                const char * contextMessage,
                                const char * filename,
                                const char * function,
                                UINT32 lineNumber
                            ) const [virtual]
```

Set the current error information associated from the nivision Imaq API.

#### Parameters:

<b>success</b>	The return from the function
<b>contextMessage</b>	A custom message from the code that set the error.
<b>filename</b>	Filename of the error source
<b>function</b>	Function of the error source
<b>lineNumber</b>	Line number of the error source

```
void ErrorBase::SetWPIError ( const char * errorMessage,
                                const char * contextMessage,
                                const char * filename,
                                const char * function,
                                UINT32 lineNumber
                            ) const [virtual]
```

Set the current error information associated with this sensor.

#### Parameters:

<b>errorMessage</b>	The error message from <a href="#">WPISensors.h</a>
<b>contextMessage</b>	A custom message from the code that set the error.
<b>filename</b>	Filename of the error source
<b>function</b>	Function of the error source
<b>lineNumber</b>	Line number of the error source

```
bool ErrorBase::StatusIsFatal ( ) const [virtual]
```

Check if the current error code represents a fatal error.

## Returns:

true if the current error is fatal.

---

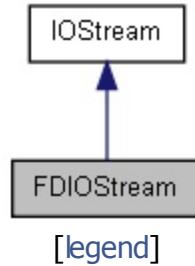
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/**ErrorBase.h**
- C:/WindRiver/workspace/WPILib/ErrorBase.cpp

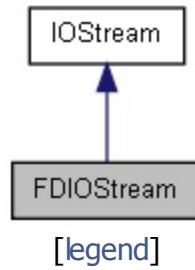


# FDIOutputStream Class Reference

Inheritance diagram for FDIOutputStream:



Collaboration diagram for FDIOutputStream:



List of all members.

# Public Member Functions

---

<b>FDIOTStream</b> (int fd)	
int <b>read</b> (void *ptr, int numbytes)	
int <b>write</b> (const void *ptr, int numbytes)	
void <b>flush</b> ()	
void <b>close</b> ()	

---

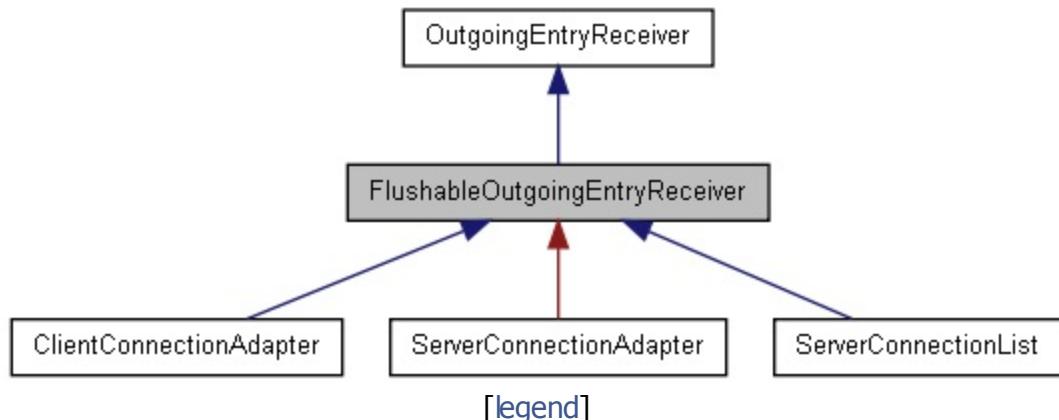
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/stream/[FDIOTStream.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/stream/FDIOTStream.cpp

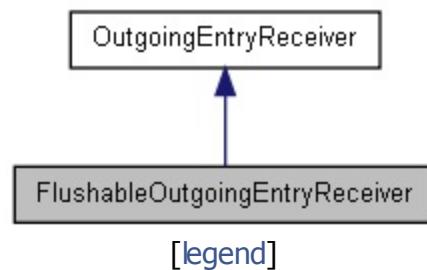


# FlushableOutgoingEntryReceiver Class Reference

Inheritance diagram for FlushableOutgoingEntryReceiver:



Collaboration diagram for FlushableOutgoingEntryReceiver:



[List of all members.](#)

# Public Member Functions

---

```
virtual void flush ()=0  
virtual void ensureAlive ()=0
```

---

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

- C:/WindRiver/workspace/WPILib/networktables2/[FlushableOutgoingEntryReceiv](#)

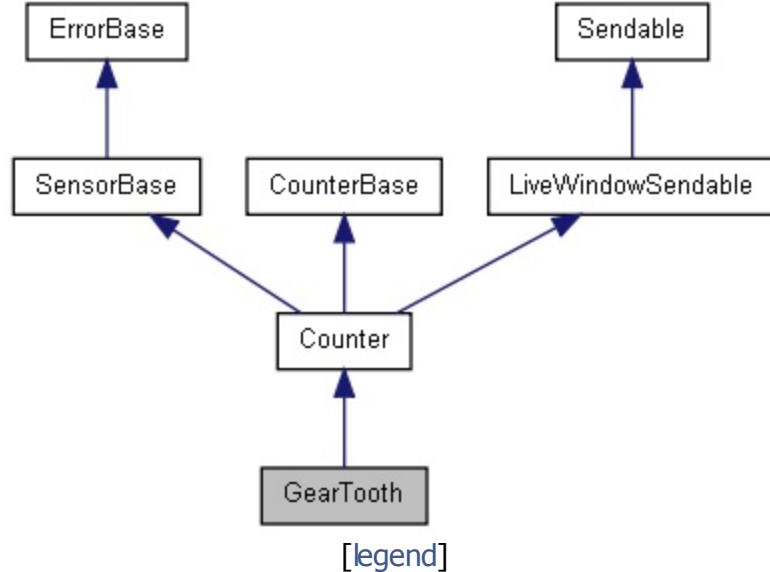


# GearTooth Class Reference

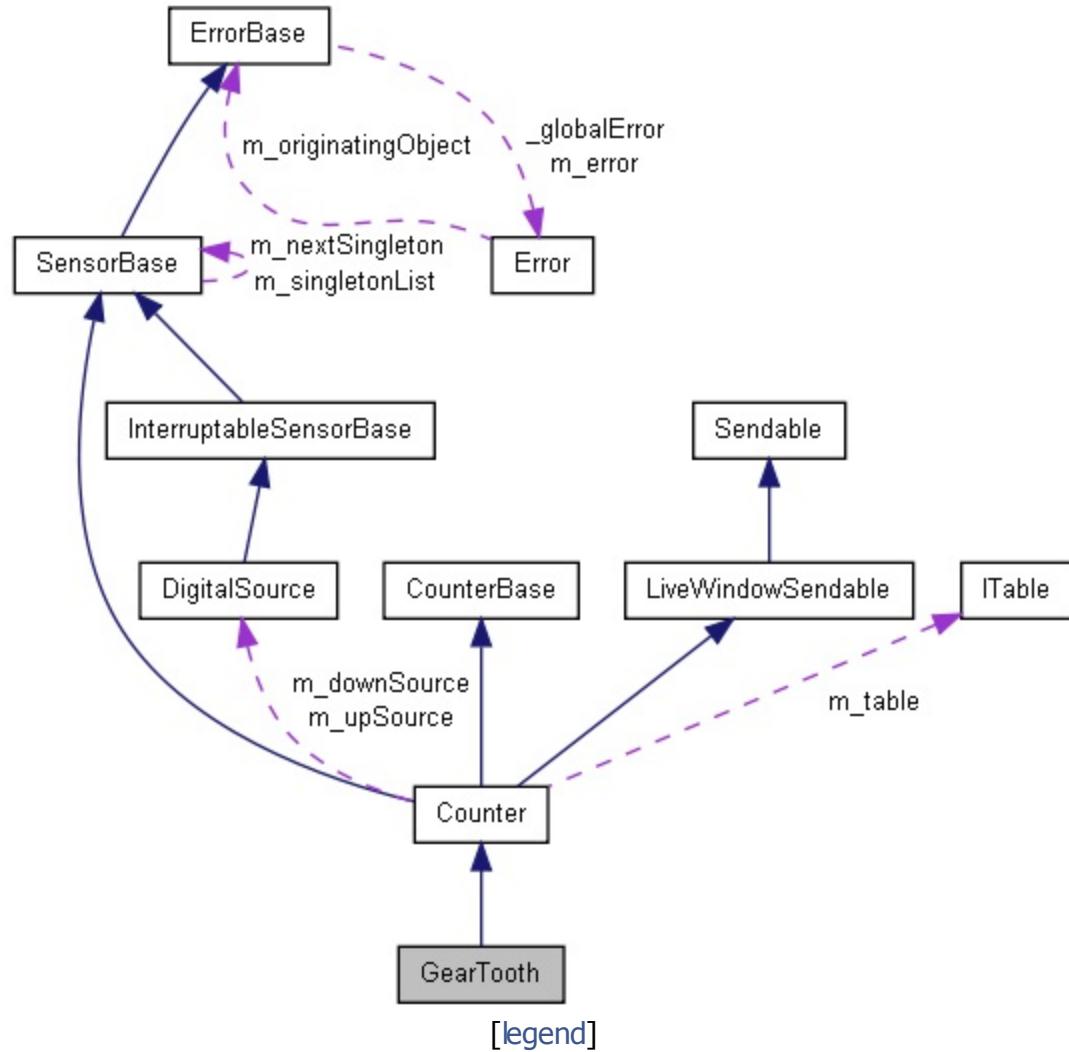
Alias for counter class. More...

```
#include <GearTooth.h>
```

Inheritance diagram for GearTooth:



Collaboration diagram for GearTooth:



List of all members.

## Public Member Functions

**GearTooth** (UINT32 channel, bool directionSensitive=false)  
Construct a **GearTooth** sensor given a channel.

**GearTooth** (UINT8 moduleNumber, UINT32 channel, bool directionSensitive=false)

Construct a **GearTooth** sensor given a channel and module.

**GearTooth** (**DigitalSource** \*source, bool directionSensitive=false)

Construct a **GearTooth** sensor given a digital input.

**GearTooth** (**DigitalSource** &source, bool directionSensitive=false)

virtual **~GearTooth** ()

Free the resources associated with a gear tooth sensor.

void **EnableDirectionSensing** (bool directionSensitive)

Common code called by the constructors.

virtual std::string **GetSmartDashboardType** ()

```
static const double kGearToothThreshold = 55e-6  
55 usec for threshold
```

---

## Detailed Description

Alias for counter class.

Implement the gear tooth sensor supplied by FIRST. Currently there is no reverse sensing on the gear tooth sensor, but in future versions we might implement the necessary timing in the FPGA to sense direction.

---

# Constructor & Destructor Documentation

```
GearTooth::GearTooth ( UINT32 channel,  
                      bool      directionSensitive = false  
)
```

Construct a **GearTooth** sensor given a channel.

The default module is assumed.

## Parameters:

<b>channel</b>	The GPIO channel on the digital module that the sensor is connected to.
<b>directionSensitive</b>	Enable the pulse length decoding in hardware to specify count direction.

```
GearTooth::GearTooth ( UINT8 moduleNumber,  
                      UINT32 channel,  
                      bool      directionSensitive = false  
)
```

Construct a **GearTooth** sensor given a channel and module.

## Parameters:

<b>moduleNumber</b>	The digital module (1 or 2).
<b>channel</b>	The GPIO channel on the digital module that the sensor is connected to.
<b>directionSensitive</b>	Enable the pulse length decoding in hardware to specify count direction.

```
GearTooth::GearTooth ( DigitalSource * source,  
                      bool           directionSensitive = false  
)
```

Construct a **GearTooth** sensor given a digital input.

This should be used when sharing digital inputs.

## Parameters:

**source** An object that fully describes the input that the sensor is connected to.

**directionSensitive** Enable the pulse length decoding in hardware to specify count direction.

---

# Member Function Documentation

`std::string GearTooth::GetSmartDashboardType( ) [virtual]`

## Returns:

the string representation of the named data type that will be used by the smart dashboard for this sendable

Reimplemented from **Counter**.

---

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

- C:/WindRiver/workspace/WPILib/**GearTooth.h**
- C:/WindRiver/workspace/WPILib/GearTooth.cpp

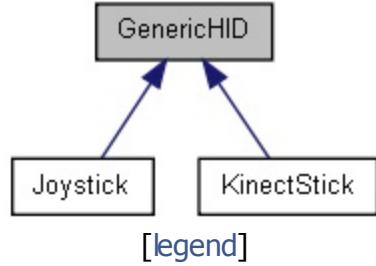


# GenericHID Class Reference

**GenericHID** Interface. More...

```
#include <GenericHID.h>
```

Inheritance diagram for GenericHID:



List of all members.

## Public Types

```
enum JoystickHand { kLeftHand = 0, kRightHand = 1 }
```

```
virtual float GetX (JoystickHand hand=kRightHand)=0
```

```
virtual float GetY (JoystickHand hand=kRightHand)=0
```

```
virtual float GetZ ()=0
```

```
virtual float GetTwist ()=0
```

```
virtual float GetThrottle ()=0
```

```
virtual float GetRawAxis (UINT32 axis)=0
```

```
virtual bool GetTrigger (JoystickHand hand=kRightHand)=0
```

```
virtual bool GetTop (JoystickHand hand=kRightHand)=0
```

```
virtual bool GetBumper (JoystickHand hand=kRightHand)=0
```

```
virtual bool GetRawButton (UINT32 button)=0
```

# Detailed Description

## GenericHID Interface.

---

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

- C:/WindRiver/workspace/WPILib/**GenericHID.h**

Generated by  1.7.2

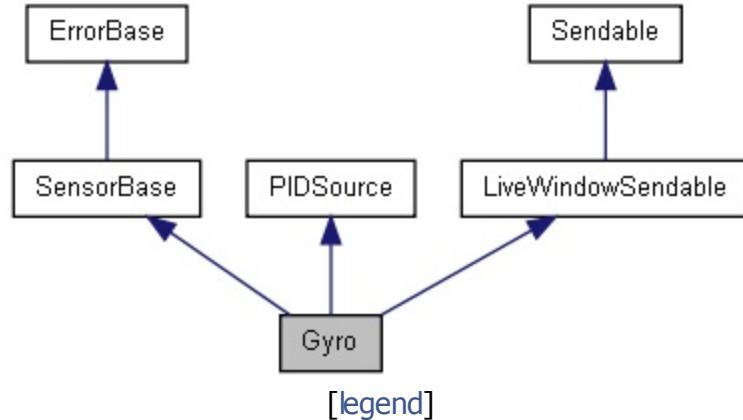


# Gyro Class Reference

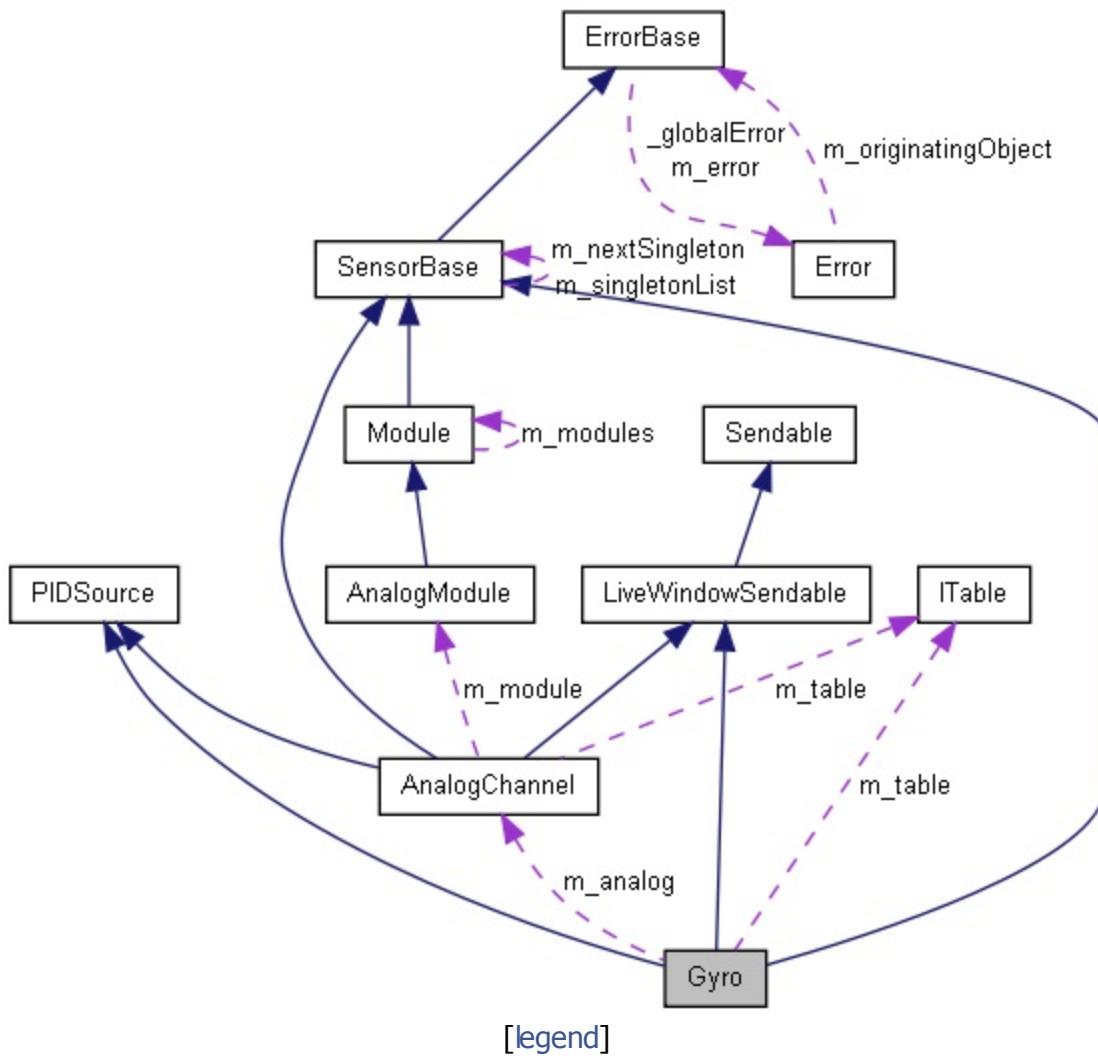
Use a rate gyro to return the robots heading relative to a starting position. More...

```
#include <Gyro.h>
```

## Inheritance diagram for Gyro:



## Collaboration diagram for Gyro:



## List of all members.

# Public Member Functions

**Gyro** (UINT8 moduleNumber, UINT32 channel)

**Gyro** constructor given a slot and a channel.

**Gyro** (UINT32 channel)

**Gyro** constructor with only a channel.

**Gyro** (**AnalogChannel** \*channel)

**Gyro** constructor with a precreated analog channel object.

**Gyro** (**AnalogChannel** &channel)

virtual **~Gyro** ()

Delete (free) the accumulator and the analog components used for the gyro.

virtual float **GetAngle** ()

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

void **SetSensitivity** (float voltsPerDegreePerSecond)

Set the gyro type based on the sensitivity.

virtual void **Reset** ()

Reset the gyro.

double **PIDGet** ()

Get the angle in degrees for the **PIDSource** base object.

void **UpdateTable** ()

Update the table for this sendable object with the latest values.

void **StartLiveWindowMode** ()

Start having this sendable object automatically respond to value changes reflect the value on the table.

void **StopLiveWindowMode** ()

Stop having this sendable object automatically respond to value changes.

std::string **GetSmartDashboardType** ()

void **InitTable** (**ITable** \*subTable)

Initializes a table for this sendable object.

**ITable** \* **GetTable** ()

## Static Public Attributes

```
static const UINT32 kOversampleBits = 10
static const UINT32 kAverageBits = 0
    static const float kSamplesPerSecond = 50.0
    static const float kCalibrationSampleTime = 5.0
    static const float kDefaultVoltsPerDegreePerSecond = 0.007
```

## Detailed Description

Use a rate gyro to return the robots heading relative to a starting position.

The [Gyro](#) class tracks the robots heading based on the starting position. As the robot rotates the new heading is computed by integrating the rate of rotation returned by the sensor. When the class is instantiated, it does a short calibration routine where it samples the gyro while at rest to determine the default offset. This is subtracted from each sample to determine the heading. This gyro class must be used with a channel that is assigned one of the Analog accumulators from the FPGA. See [AnalogChannel](#) for the current accumulator assignments.

---

# Constructor & Destructor Documentation

```
Gyro::Gyro( UINT8 moduleNumber,  
            UINT32 channel  
        )
```

**Gyro** constructor given a slot and a channel.

## Parameters:

**moduleNumber** The analog module the gyro is connected to (1).  
**channel** The analog channel the gyro is connected to (1 or 2).

```
Gyro::Gyro( UINT32 channel ) [explicit]
```

**Gyro** constructor with only a channel.

Use the default analog module slot.

## Parameters:

**channel** The analog channel the gyro is connected to.

```
Gyro::Gyro( AnalogChannel * channel ) [explicit]
```

**Gyro** constructor with a precreated analog channel object.

Use this constructor when the analog channel needs to be shared. There is no reference counting when an **AnalogChannel** is passed to the gyro.

## Parameters:

**channel** The **AnalogChannel** object that the gyro is connected to.

# Member Function Documentation

## **float Gyro::GetAngle( void ) [virtual]**

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.

### **Returns:**

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

## **std::string Gyro::GetSmartDashboardType( ) [virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

## **ITable \* Gyro::GetTable( ) [virtual]**

### **Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

## **void Gyro::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

## **double Gyro::PIDGet( ) [virtual]**

Get the angle in degrees for the **PIDSource** base object.

### **Returns:**

The angle in degrees.

Implements **PIDSource**.

## **void Gyro::Reset( ) [virtual]**

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.

## **void Gyro::SetSensitivity ( 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:**

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

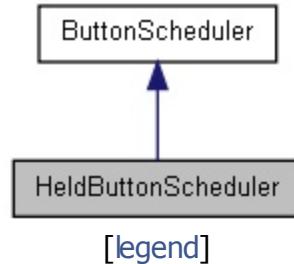
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPIlib/**Gyro.h**
- C:/WindRiver/workspace/WPIlib/Gyro.cpp

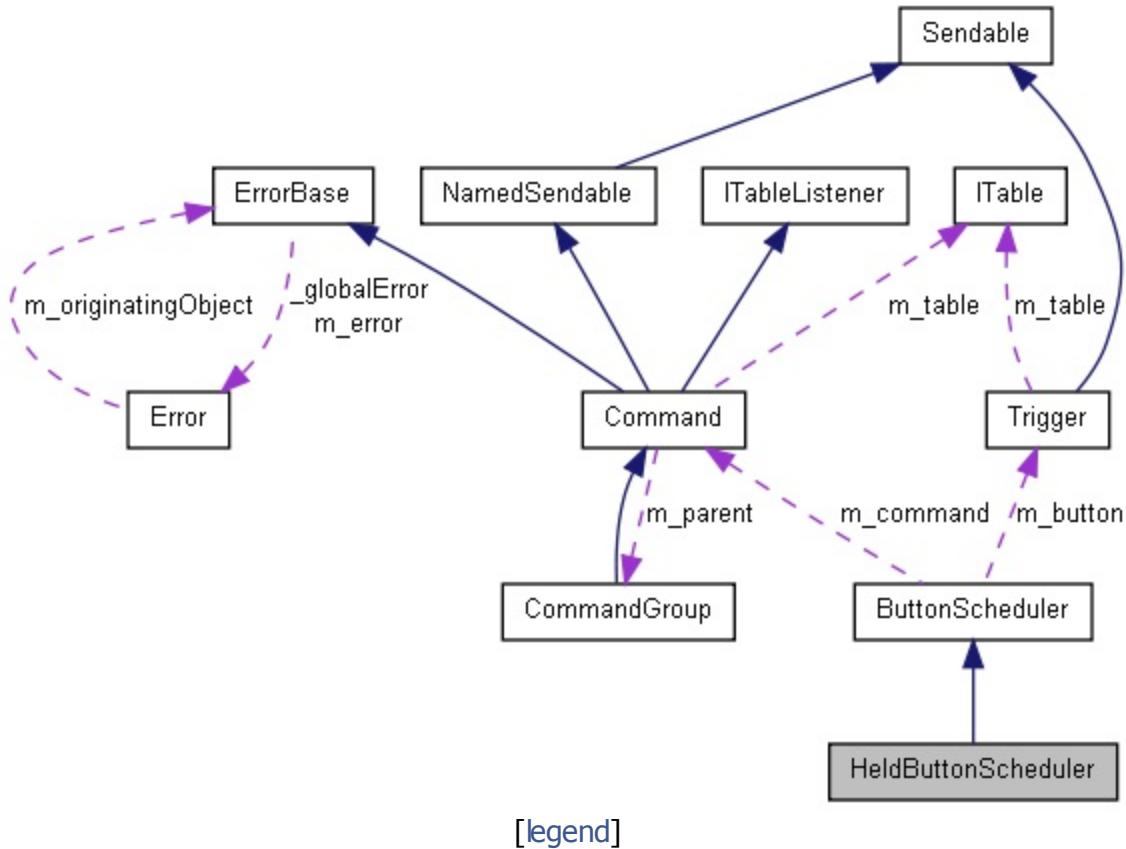


# HeldButtonScheduler Class Reference

## Inheritance diagram for HeldButtonScheduler:



## Collaboration diagram for HeldButtonScheduler:



# Public Member Functions

---

**HeldButtonScheduler** (bool last, **Trigger** \*button, **Command** \*orders)

virtual void **Execute** ()

---

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

- C:/WindRiver/workspace/WPILib/Buttons/**HeldButtonScheduler.h**
- C:/WindRiver/workspace/WPILib/Buttons/HeldButtonScheduler.cpp

Generated by  1.7.2

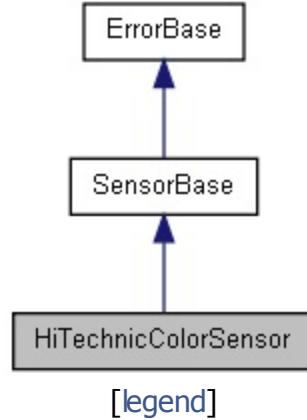


# HiTechnicColorSensor Class Reference

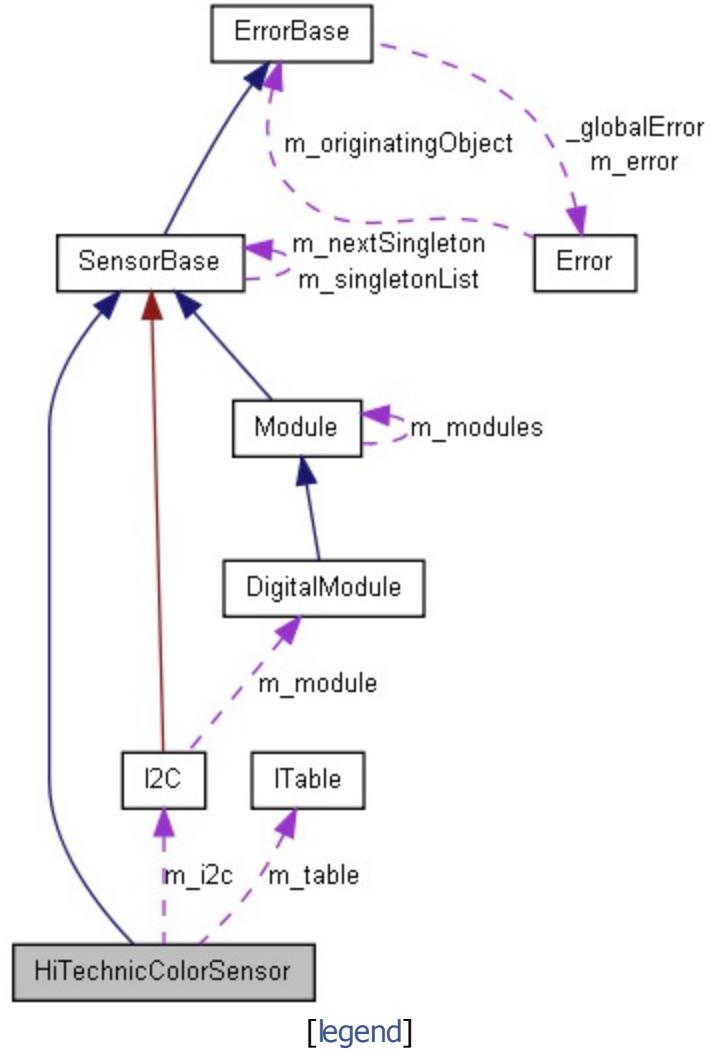
HiTechnic NXT Color Sensor. More...

```
#include <HiTechnicColorSensor.h>
```

Inheritance diagram for HiTechnicColorSensor:



Collaboration diagram for HiTechnicColorSensor:



List of all members.

# Classes

struct **RGB**

## Public Types

```
enum tColorMode { kActive = 0, kPassive = 1, kRaw = 3 }
```

## **HiTechnicColorSensor** (UINT8 moduleNumber)

Constructor.

virtual **~HiTechnicColorSensor** ()

Destructor.

UINT8 **GetColor** ()

Get the estimated color.

UINT8 **GetRed** ()

Get the Red value.

UINT8 **GetGreen** ()

Get the Green value.

UINT8 **GetBlue** ()

Get the Blue value.

**RGB** **GetRGB** ()

Get the value of all three colors from a single sensor reading.

UINT16 **GetRawRed** ()

Get the Raw Red value.

UINT16 **GetRawGreen** ()

Get the Raw Green value.

UINT16 **GetRawBlue** ()

Get the Raw Blue value.

**RGB** **GetRawRGB** ()

Get the raw value of all three colors from a single sensor reading.

void **SetMode** (tColorMode mode)

Set the Mode of the color sensor This method is used to set the color sensor to one of the three modes, active, passive or raw.

virtual std::string **GetType** ()

virtual void **InitTable** (ITable \*subtable)

virtual void **UpdateTable** ()

virtual ITable \* **GetTable** ()

virtual void **StartLiveWindowMode** ()

virtual void **StopLiveWindowMode** ()

## Detailed Description

HiTechnic NXT Color Sensor.

This class allows access to a HiTechnic NXT Color Sensor on an **I2C** bus. These sensors do not allow changing addresses so you cannot have more than one on a single bus.

Details on the sensor can be found here: <http://www.hitechnic.com/index.html?lang=en-us&target=d17.html>

---

# Constructor & Destructor Documentation

## HiTechnicColorSensor::HiTechnicColorSensor ( **UINT8 moduleNumber** ) [explicit]

Constructor.

### Parameters:

**moduleNumber** The digital module that the sensor is plugged into (1 or 2).

# Member Function Documentation

## **UINT8 HiTechnicColorSensor::GetBlue( )**

Get the Blue value.

Gets the raw (0-255) blue value from the sensor.

The sensor must be in active mode to access the regular **RGB** data if the sensor is not in active mode, it will be placed into active mode by this method.

### **Returns:**

The Blue sensor value.

## **UINT8 HiTechnicColorSensor::GetColor( )**

Get the estimated color.

Gets a color estimate from the sensor corresponding to the table found with the sensor or at the following site: <http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NCO1038>

### **Returns:**

The estimated color.

## **UINT8 HiTechnicColorSensor::GetGreen( )**

Get the Green value.

Gets the(0-255) green value from the sensor.

The sensor must be in active mode to access the regular **RGB** data if the sensor is not in active mode, it will be placed into active mode by this method.

### **Returns:**

The Green sensor value.

## **UINT16 HiTechnicColorSensor::GetRawBlue( )**

Get the Raw Blue value.

Gets the (0-65536) raw blue value from the sensor.

The sensor must be in raw or passive mode to access the regular **RGB** data if the sensor is not in raw or passive mode, it will be placed into raw mode by this method.

**Returns:**

The Raw Blue sensor value.

## **UINT16 HiTechnicColorSensor::GetRawGreen( )**

Get the Raw Green value.

Gets the (0-65536) raw green value from the sensor.

The sensor must be in raw or passive mode to access the regular **RGB** data if the sensor is not in raw or passive mode, it will be placed into raw mode by this method.

**Returns:**

The Raw Green sensor value.

## **UINT16 HiTechnicColorSensor::GetRawRed( )**

Get the Raw Red value.

Gets the (0-65536) raw red value from the sensor.

The sensor must be in raw or passive mode to access the regular **RGB** data if the sensor is not in raw or passive mode, it will be placed into raw mode by this method.

**Returns:**

The Raw Red sensor value.

## **HiTechnicColorSensor::RGB HiTechnicColorSensor::GetRawRGB( )**

Get the raw value of all three colors from a single sensor reading.

Using this method ensures that all three values come from the same sensor reading, using the individual color methods provides no such guarantee.

Gets the (0-65536) raw color values from the sensor.

The sensor must be in raw or passive mode to access the regular **RGB** data if the

sensor is not in raw or passive mode, it will be placed into raw mode by this method.

### Returns:

An **RGB** object with the raw sensor values.

## **UINT8 HiTechnicColorSensor::GetRed( )**

Get the Red value.

Gets the (0-255) red value from the sensor.

The sensor must be in active mode to access the regular **RGB** data if the sensor is not in active mode, it will be placed into active mode by this method.

### Returns:

The Red sensor value.

## **HiTechnicColorSensor::RGB HiTechnicColorSensor::GetRGB( )**

Get the value of all three colors from a single sensor reading.

Using this method ensures that all three values come from the same sensor reading, using the individual color methods provides no such guarantee.

The sensor must be in active mode to access the regular **RGB** data. If the sensor is not in active mode, it will be placed into active mode by this method.

### Returns:

**RGB** object with the three color values

## **void HiTechnicColorSensor::SetMode( tColorMode mode )**

Set the Mode of the color sensor This method is used to set the color sensor to one of the three modes, active, passive or raw.

The sensor defaults to active mode which uses the internal LED and returns an interpreted color value and 3 8-bit **RGB** channel values. Raw mode uses the internal LED and returns 3 16-bit **RGB** channel values. Passive mode disables the internal LED and returns 3 16-bit **RGB** channel values.

### Parameters:

## **mode** The mode to set

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

- C:/WindRiver/workspace/WPILib/**HiTechnicColorSensor.h**
- C:/WindRiver/workspace/WPILib/HiTechnicColorSensor.cpp

Generated by  1.7.2

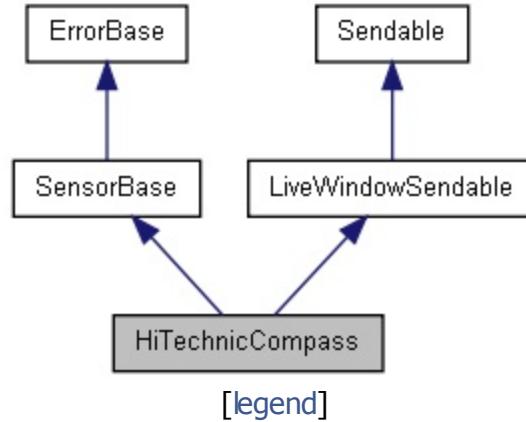


# HiTechnicCompass Class Reference

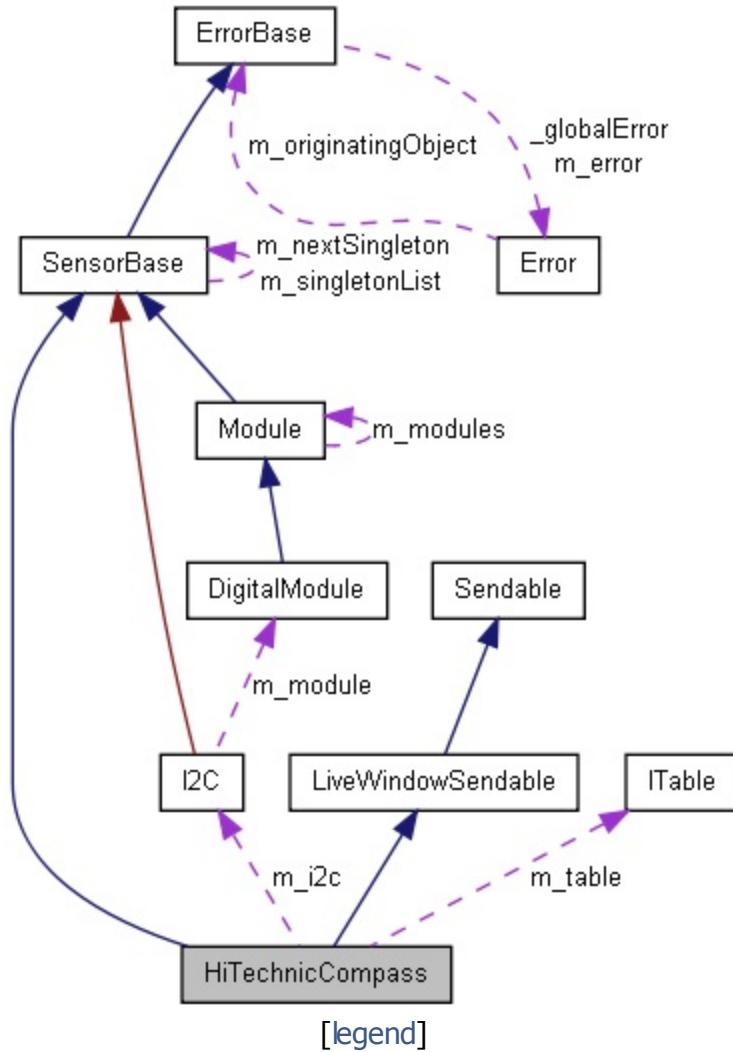
HiTechnic NXT Compass. More...

```
#include <HiTechnicCompass.h>
```

Inheritance diagram for HiTechnicCompass:



Collaboration diagram for HiTechnicCompass:



List of all members.

# Public Member Functions

**HiTechnicCompass** (UINT8 moduleNumber)

Constructor.

virtual **~HiTechnicCompass** ()

Destructor.

float **GetAngle** ()

Get the compass angle in degrees.

void **UpdateTable** ()

Update the table for this sendable object with the latest values.

void **StartLiveWindowMode** ()

Start having this sendable object automatically respond to value changes reflect the value on the table.

void **StopLiveWindowMode** ()

Stop having this sendable object automatically respond to value changes.

std::string **GetSmartDashboardType** ()

void **InitTable** (ITable \*subTable)

Initializes a table for this sendable object.

**ITable** \* **GetTable** ()

# Detailed Description

HiTechnic NXT Compass.

This class allows access to a HiTechnic NXT Compass on an **I2C** bus. These sensors do not allow changing addresses so you cannot have more than one on a single bus.

Details on the sensor can be found here: <http://www.hitechnic.com/index.html?lang=en-us&target=d17.html>

## **Todo:**

Implement a calibration method for the sensor.

---

# Constructor & Destructor Documentation

## HiTechnicCompass::HiTechnicCompass ( **UINT8 moduleNumber** ) [explicit]

Constructor.

### Parameters:

**moduleNumber** The digital module that the sensor is plugged into (1 or 2).

# Member Function Documentation

## **float HiTechnicCompass::GetAngle( void )**

Get the compass angle in degrees.

The resolution of this reading is 1 degree.

### Returns:

Angle of the compass in degrees.

## **std::string HiTechnicCompass::GetSmartDashboardType( ) [virtual]**

### Returns:

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

## **ITable \* HiTechnicCompass::GetTable( ) [virtual]**

### Returns:

the table that is currently associated with the sendable

Implements **Sendable**.

## **void HiTechnicCompass::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

### Parameters:

**subtable** The table to put the values in.

Implements **Sendable**.

---

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

- C:/WindRiver/workspace/WPILib/**HiTechnicCompass.h**
- C:/WindRiver/workspace/WPILib/HiTechnicCompass.cpp



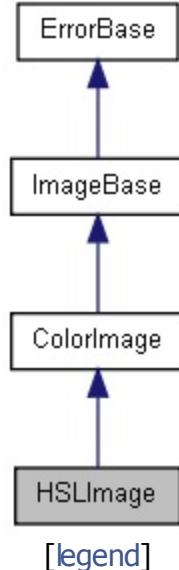


# HSLImage Class Reference

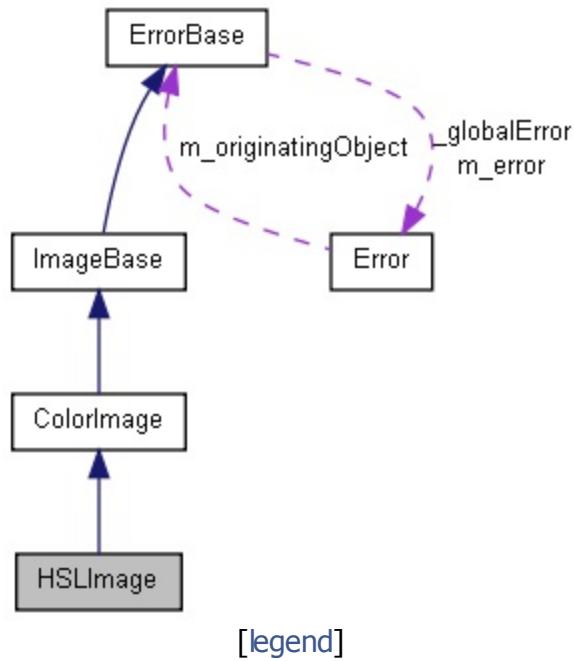
A color image represented in HSL color space at 3 bytes per pixel. [More...](#)

```
#include <HSLImage.h>
```

Inheritance diagram for HSLImage:



Collaboration diagram for HSLImage:



[List of all members.](#)

## Public Member Functions

### **HSLImage ()**

Create a new image that uses the Hue, Saturation, and Luminance planes.

### **HSLImage (const char \*fileName)**

Create a new image by loading a file.

## Detailed Description

A color image represented in HSL color space at 3 bytes per pixel.

---

# Constructor & Destructor Documentation

## **HSLImage::HSLImage ( const char \* fileName )**

Create a new image by loading a file.

### **Parameters:**

**fileName** The path of the file to load.

---

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

- C:/WindRiver/workspace/WPILib/Vision/**HSLImage.h**
- C:/WindRiver/workspace/WPILib/Vision/HSLImage.cpp

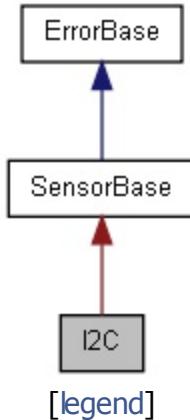


# I2C Class Reference

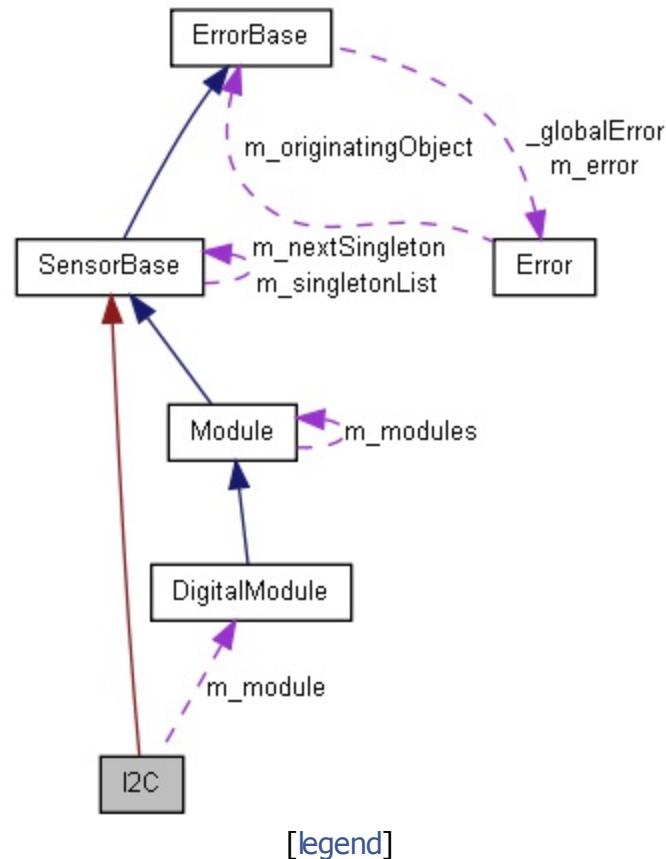
I2C bus interface class. More...

```
#include <I2C.h>
```

Inheritance diagram for I2C:



Collaboration diagram for I2C:



List of all members.

# Public Member Functions

virtual	<b><code>~I2C ()</code></b>	Destructor.
bool	<b><code>Transaction (UINT8 *dataToSend, UINT8 sendSize, UINT8 *dataReceived, UINT8 receiveSize)</code></b>	Generic transaction.
bool	<b><code>AddressOnly ()</code></b>	Attempt to address a device on the <b>I2C</b> bus.
bool	<b><code>Write (UINT8 registerAddress, UINT8 data)</code></b>	Execute a write transaction with the device.
bool	<b><code>Read (UINT8 registerAddress, UINT8 count, UINT8 *data)</code></b>	Execute a read transaction with the device.
void	<b><code>Broadcast (UINT8 registerAddress, UINT8 data)</code></b>	Send a broadcast write to all devices on the <b>I2C</b> bus.
void	<b><code>SetCompatibilityMode (bool enable)</code></b>	SetCompatibilityMode.
bool	<b><code>VerifySensor (UINT8 registerAddress, UINT8 count, const UINT8 *expected)</code></b>	Verify that a device's registers contain expected values.

# Friends

class **DigitalModule**

---

## Detailed Description

**I2C** bus interface class.

This class is intended to be used by sensor (and other **I2C** device) drivers. It probably should not be used directly.

It is constructed by calling **DigitalModule::GetI2C()** on a **DigitalModule** object.

---

# Member Function Documentation

## **bool I2C::AddressOnly ( )**

Attempt to address a device on the **I2C** bus.

This allows you to figure out if there is a device on the **I2C** bus that responds to the address specified in the constructor.

### **Returns:**

Transfer Aborted... false for success, true for aborted.

## **void I2C::Broadcast ( UINT8 registerAddress,                         UINT8 data                         )**

Send a broadcast write to all devices on the **I2C** bus.

This is not currently implemented!

### **Parameters:**

**registerAddress** The register to write on all devices on the bus.  
**data** The value to write to the devices.

## **bool I2C::Read ( UINT8 registerAddress,                     UINT8 count,                     UINT8 \* buffer                     )**

Execute a read transaction with the device.

Read 1 to 7 bytes from a device. Most **I2C** devices will auto-increment the register pointer internally allowing you to read up to 7 consecutive registers on a device in a single transaction.

### **Parameters:**

**registerAddress** The register to read first in the transaction.  
**count** The number of bytes to read in the transaction. [1..7]  
**buffer** A pointer to the array of bytes to store the data read from the device.

## Returns:

Transfer Aborted... false for success, true for aborted.

### **void I2C::SetCompatibilityMode ( bool enable )**

SetCompatibilityMode.

Enables bitwise clock skewing detection. This will reduce the **I2C** interface speed, but will allow you to communicate with devices that skew the clock at abnormal times.

## Parameters:

**enable** Enable compatibility mode for this sensor or not.

### **bool I2C::Transaction ( UINT8 \* dataToSend,                         UINT8 sendSize,                         UINT8 \* dataReceived,                         UINT8 receiveSize                         )**

Generic transaction.

This is a lower-level interface to the **I2C** hardware giving you more control over each transaction.

## Parameters:

**dataToSend** Buffer of data to send as part of the transaction.

**sendSize** Number of bytes to send as part of the transaction. [0..6]

**dataReceived** Buffer to read data into.

**receiveSize** Number of bytes to read from the device. [0..7]

## Returns:

Transfer Aborted... false for success, true for aborted.

### **bool I2C::VerifySensor ( UINT8 registerAddress,                         UINT8 count,                         const UINT8 \* expected                         )**

Verify that a device's registers contain expected values.

Most devices will have a set of registers that contain a known value that can be used to identify them. This allows an **I2C** device driver to easily verify that the device contains the expected value.

### Precondition:

The device must support and be configured to use register auto-increment.

### Parameters:

**registerAddress** The base register to start reading from the device.

**count** The size of the field to be verified.

**expected** A buffer containing the values expected from the device.

```
bool I2C::Write ( UINT8 registerAddress,  
                   UINT8 data  
)
```

Execute a write transaction with the device.

Write a single byte to a register on a device and wait until the transaction is complete.

### Parameters:

**registerAddress** The address of the register on the device to be written.

**data** The byte to write to the register on the device.

### Returns:

Transfer Aborted... false for success, true for aborted.

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

- C:/WindRiver/workspace/WPIlib/**I2C.h**
- C:/WindRiver/workspace/WPIlib/I2C.cpp



# IllegalStateException Class Reference

---

List of all members.

# Public Member Functions

**IllegalStateException** (const char \*message)

const char \* **what** ()

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

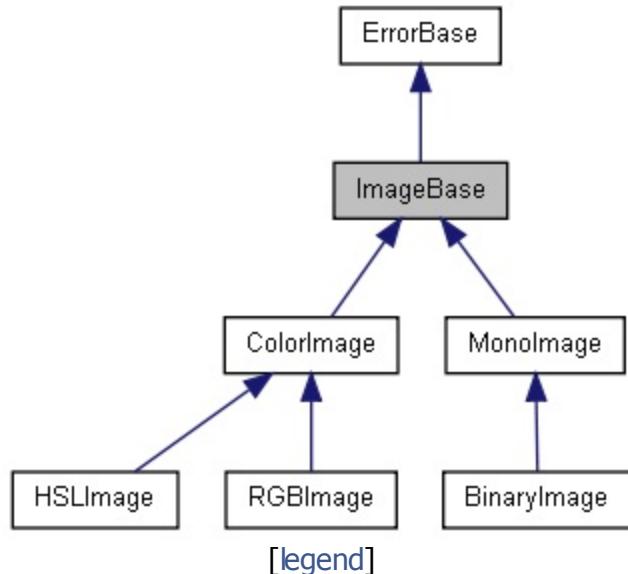
- C:/WindRiver/workspace/WPILib/networktables2/util/[IllegalStateException.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/util/IllegalStateException.cpp

Generated by  1.7.2

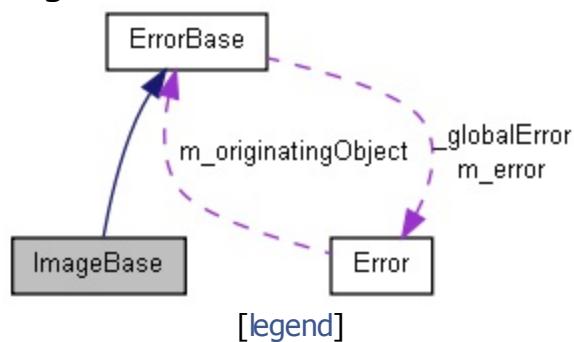


# ImageBase Class Reference

Inheritance diagram for ImageBase:



Collaboration diagram for ImageBase:



[List of all members.](#)

# Public Member Functions

**ImageBase** (ImageType type)

Create a new instance of an **ImageBase**.

virtual **~ImageBase** ()

Frees memory associated with an **ImageBase**.

virtual void **Write** (const char \*fileName)

Writes an image to a file with the given filename.

int **GetHeight** ()

Gets the height of an image.

int **GetWidth** ()

Gets the width of an image.

Image \* **GetImaqImage** ()

Access the internal IMAQ Image data structure.

Image \* **m\_imaqImage**

# Constructor & Destructor Documentation

## **ImageBase::ImageBase ( ImageType type )**

Create a new instance of an **ImageBase**.

Imagebase is the base of all the other image classes. The constructor creates any type of image and stores the pointer to it in the class.

### **Parameters:**

**type** The type of image to create

## **ImageBase::~ImageBase ( ) [virtual]**

Frees memory associated with an **ImageBase**.

Destructor frees the imaq image allocated with the class.

# Member Function Documentation

## **int ImageBase::GetHeight( )**

Gets the height of an image.

### **Returns:**

The height of the image in pixels.

## **Image \* ImageBase::GetImaqImage( )**

Access the internal IMAQ Image data structure.

### **Returns:**

A pointer to the internal IMAQ Image data structure.

## **int ImageBase::GetWidth( )**

Gets the width of an image.

### **Returns:**

The width of the image in pixels.

## **void ImageBase::Write( const char \* fileName ) [virtual]**

Writes an image to a file with the given filename.

Write the image to a file in the flash on the cRIO.

### **Parameters:**

**fileName** The name of the file to write

Reimplemented in [BinaryImage](#).

---

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

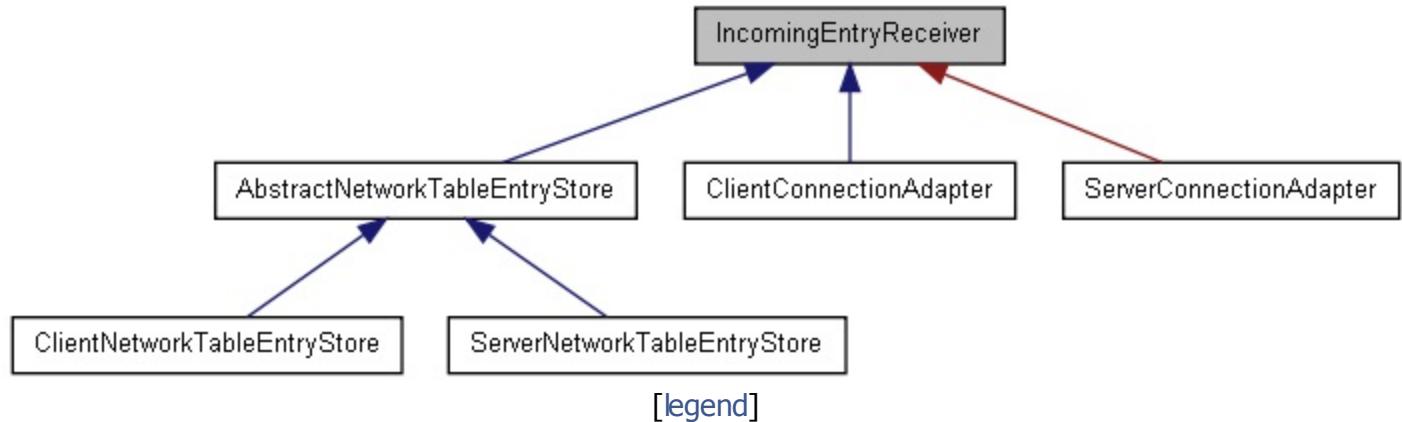
- C:/WindRiver/workspace/WPILib/Vision/[ImageBase.h](#)
- C:/WindRiver/workspace/WPILib/Vision/ImageBase.cpp





# IncomingEntryReceiver Class Reference

Inheritance diagram for IncomingEntryReceiver:



List of all members.

## Public Member Functions

---

```
virtual void offerIncomingAssignment (NetworkTableEntry *entry)=0
```

```
virtual void offerIncomingUpdate (NetworkTableEntry *entry, SequenceNumber  
entrySequenceNumber, EntryValue value)=0
```

---

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

- C:/WindRiver/workspace/WPILib/networktables2/**IncomingEntryReceiver.h**

[Class List](#)[Class Hierarchy](#)[Class Members](#)

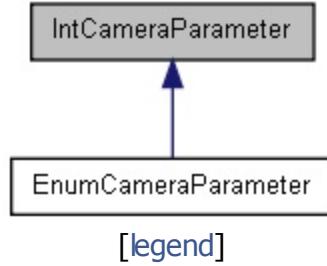
[Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#)

# IntCameraParameter Class Reference

Integer camera parameter. [More...](#)

```
#include <IntCameraParameter.h>
```

Inheritance diagram for IntCameraParameter:



[List of all members.](#)

# Public Member Functions

**IntCameraParameter** (const char \*setString, const char \*getString,  
bool requiresRestart)  
Constructor for an integer camera parameter.

int **GetValue** ()  
Get a value for a camera parameter.

void **SetValue** (int value)  
Set a value for a camera parameter.

virtual bool **CheckChanged** (bool &changed, char \*param)  
Check if a parameter has changed and update.

virtual void **GetParamFromString** (const char \*string, int stringLength)  
Get a parameter value from the string.

int **SearchForParam** (const char \*pattern, const char \*searchString, int  
searchStringLen, char \*result)

const char \* **m\_setString**

```
const char * mGetString  
bool m_changed  
bool m_requiresRestart  
int m_value
```

---

## Detailed Description

Integer camera parameter.

This class represents a camera parameter that takes an integer value.

---

# Constructor & Destructor Documentation

```
IntCameraParameter::IntCameraParameter ( const char * setString,  
                                         const char * getString,  
                                         bool           requiresRestart  
                                         )
```

Constructor for an integer camera parameter.

## Parameters:

- setString** The string to set a value in the HTTP request
- getString** The string to retrieve a value in the HTTP request

# Member Function Documentation

Check if a parameter has changed and update.

Check if a parameter has changed and send the update string if it has changed. This is called from the loop in the parameter task loop.

## Returns:

true if the camera needs to restart

Reimplemented in [EnumCameraParameter](#).

Get a parameter value from the string.

Get a parameter value from the camera status string. If it has been changed by the program, then don't update it. Program values have precedence over those written in the camera.

Reimplemented in [EnumCameraParameter](#).

## **int IntCameraParameter::GetValue( )**

Get a value for a camera parameter.

## Returns:

The camera parameter cached valued.

**Parameters:**

<b>pattern,:</b>	the regular expression
<b>searchString</b>	the text to search
<b>searchStringLen</b>	the length of searchString
<b>result</b>	buffer to put resulting text into, must be pre-allocated

**void IntCameraParameter::SetValue ( int value )**

Set a value for a camera parameter.

Mark the value for change. The value will be updated in the parameter change loop.

---

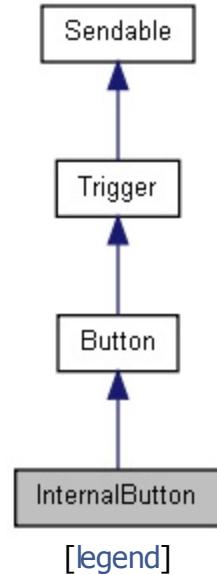
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/Vision/**IntCameraParameter.h**
- C:/WindRiver/workspace/WPILib/Vision/IntCameraParameter.cpp

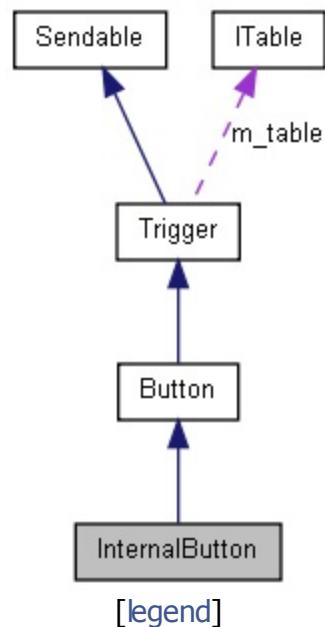


# InternalButton Class Reference

Inheritance diagram for InternalButton:



Collaboration diagram for InternalButton:



List of all members.

# Public Member Functions

**InternalButton** (bool inverted)

void **SetInverted** (bool inverted)

void **SetPressed** (bool pressed)

virtual bool **Get** ()

---

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

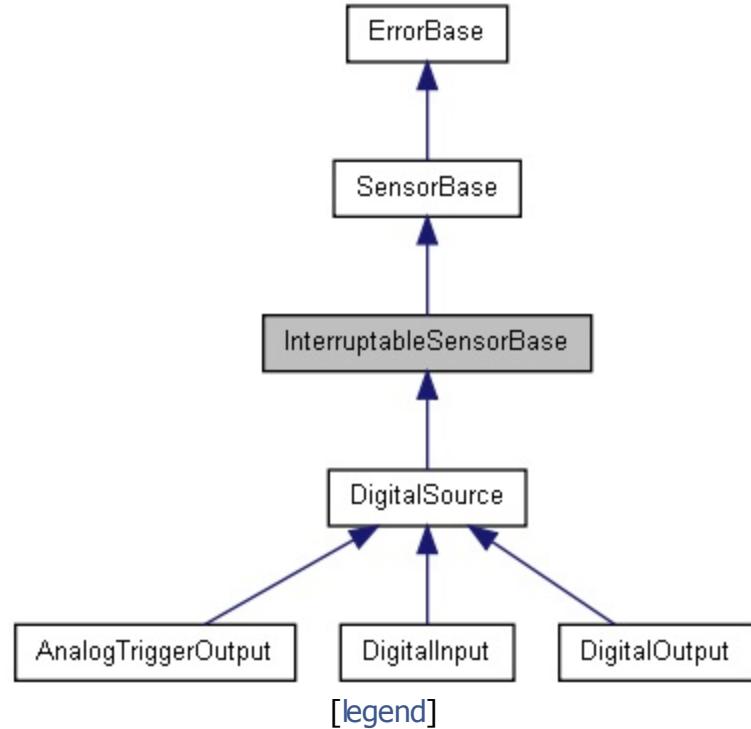
- C:/WindRiver/workspace/WPILib/Buttons/[InternalButton.h](#)
- C:/WindRiver/workspace/WPILib/Buttons/InternalButton.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

[Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#)

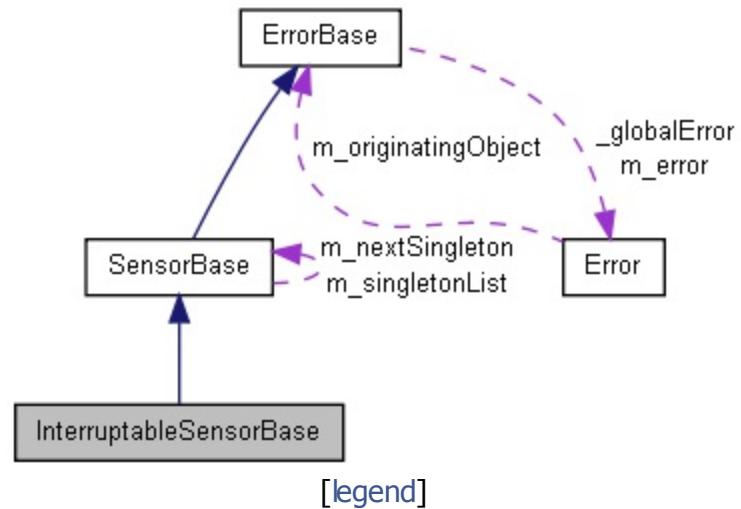
# InterruptableSensorBase Class Reference

Inheritance diagram for InterruptableSensorBase:



[legend]

Collaboration diagram for InterruptableSensorBase:



[legend]

List of all members.

## Public Member Functions

virtual void **RequestInterrupts** (tInterruptHandler handler, void \*param)=0  
Asynchronous handler version.

virtual void **RequestInterrupts** ()=0  
Synchronous Wait version.

virtual void **CancelInterrupts** ()  
Free up the underlying chipobject functions.

virtual void **WaitForInterrupt** (float timeout)  
Synchronous version.

virtual void **EnableInterrupts** ()  
Enable interrupts - after finishing setup.

virtual void **DisableInterrupts** ()  
Disable, but don't deallocate.

virtual double **ReadInterruptTimestamp** ()  
Return the timestamp for the interrupt that occurred.

# Protected Member Functions

---

```
void AllocateInterrupts (bool watcher)
```

## **Protected Attributes**

tInterrupt \* **m\_interrupt**

tInterruptManager \* **m\_manager**

UINT32 **m\_interruptIndex**

---

# Member Function Documentation

## **void InterruptableSensorBase::CancelInterrupts( ) [virtual]**

Free up the underlying chipobject functions.

Cancel interrupts on this device.

This deallocates all the chipobject structures and disables any interrupts.

## **void InterruptableSensorBase::DisableInterrupts( ) [virtual]**

Disable, but don't deallocate.

Disable Interrupts without deallocating structures.

## **void InterruptableSensorBase::EnableInterrupts( ) [virtual]**

Enable interrupts - after finishing setup.

Enable interrupts to occur on this input.

Interrupts are disabled when the RequestInterrupt call is made. This gives time to do the setup of the other options before starting to field interrupts.

## **double InterruptableSensorBase::ReadInterruptTimestamp( ) [virtual]**

Return the timestamp for the interrupt that occurred.

Return the timestamp for the interrupt that occurred most recently.

This is in the same time domain as GetClock().

### **Returns:**

Timestamp in seconds since boot.

## **void InterruptableSensorBase::WaitForInterrupt( float timeout ) [virtual]**

Synchronous version.

In synchronous mode, wait for the defined interrupt to occur.

## Parameters:

**timeout** Timeout in seconds

---

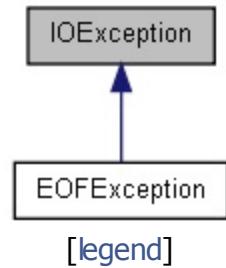
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/[InterruptableSensorBase.h](#)
- C:/WindRiver/workspace/WPILib/InterruptableSensorBase.cpp



# IOException Class Reference

Inheritance diagram for IOException:



[[legend](#)]

List of all members.

# Public Member Functions

---

**IOException** (const char \*message)  
**IOException** (const char \*message, int errorValue)  
const char \* **what** ()  
virtual bool **isEOF** ()

---

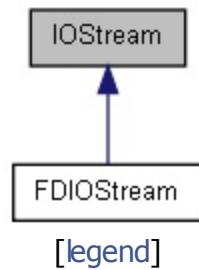
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/util/**IOException.h**
- C:/WindRiver/workspace/WPILib/networktables2/util/IOException.cpp



# IOStream Class Reference

Inheritance diagram for IOStream:



List of all members.

## Public Member Functions

---

```
virtual int read (void *ptr, int numbytes)=0
virtual int write (const void *ptr, int numbytes)=0
virtual void flush ()=0
virtual void close ()=0
```

---

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

- C:/WindRiver/workspace/WPILib/networktables2/stream/**IOStream.h**

Generated by  1.7.2

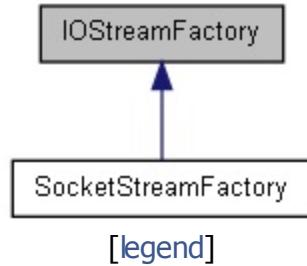


# IOStreamFactory Class Reference

A factory that will create the same **IOStream**. More...

```
#include <IOStreamFactory.h>
```

Inheritance diagram for IOStreamFactory:



List of all members.

## Public Member Functions

virtual **IOStream** \* **createStream** ()=0

---

## Detailed Description

A factory that will create the same [IOStream](#).

A stream returned by this factory should be closed before calling `createStream` again

### Author:

Mitchell

---

# Member Function Documentation

**virtual [IOStream\\*](#) [IOStreamFactory::createStream\(\)](#) [pure virtual]**

## Returns:

create a new stream

## Exceptions:

[IOException](#)

Implemented in [SocketStreamFactory](#).

---

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

- C:/WindRiver/workspace/WPILib/networktables2/stream/[IOStreamFactory.h](#)

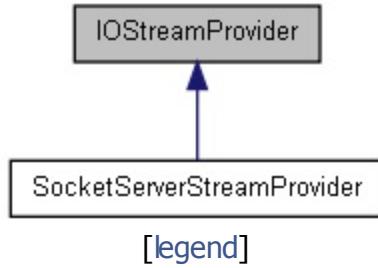


# IOStreamProvider Class Reference

An object that will provide the **IOStream** of clients to a **NetworkTable** Server. More...

```
#include <IOStreamProvider.h>
```

Inheritance diagram for IOStreamProvider:



List of all members.

## Public Member Functions

virtual **IOStream** \* **accept ()=0**

virtual void **close ()=0**

Close the source of the IOStreams.

---

## Detailed Description

An object that will provide the **IOStream** of clients to a **NetworkTable** Server.

### Author:

mwills

---

# Member Function Documentation

**virtual `IOStream*` `IOStreamProvider::accept( )` [pure virtual]**

## Returns:

a new **IOStream** normally from a server

## Exceptions:

**IOException**

Implemented in **SocketServerStreamProvider**.

**virtual void `IOStreamProvider::close( )` [pure virtual]**

Close the source of the IOStreams.

**accept()** should not be called after this is called

## Exceptions:

**IOException**

Implemented in **SocketServerStreamProvider**.

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

- C:/WindRiver/workspace/WPILib/networktables2/stream/**IOStreamProvider.h**

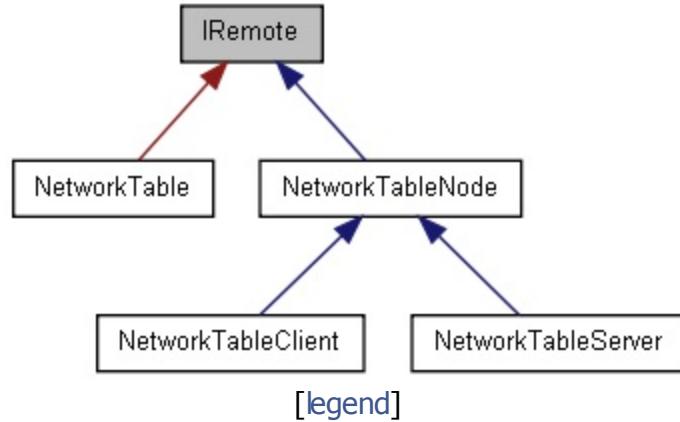


# IRemote Class Reference

Represents an object that has a remote connection. [More...](#)

```
#include <IRemote.h>
```

Inheritance diagram for IRemote:



[List of all members.](#)

## Public Member Functions

virtual void	<b>AddConnectionListener</b> ( <b>IRemoteConnectionListener</b> *listener, bool immediateNotify)=0 Register an object to listen for connection and disconnection events.
virtual void	<b>RemoveConnectionListener</b> ( <b>IRemoteConnectionListener</b> *listener)=0 Unregister a listener from connection events.
virtual bool	<b>IsConnected</b> ()=0 Get the current state of the objects connection.
virtual bool	<b>IsServer</b> ()=0 If the object is acting as a server.

## Detailed Description

Represents an object that has a remote connection.

### Author:

Mitchell

---

# Member Function Documentation

```
virtual void IRemote::AddConnectionListener ( IRemoteConnectionListener *  
                                         bool  
                                         )
```

Register an object to listen for connection and disconnection events.

## Parameters:

- listener** the listener to be register
- immediateNotify** if the listener object should be notified of the current connection state

Implemented in [NetworkTable](#), and [NetworkTableNode](#).

```
virtual bool IRemote::IsConnected ( ) [pure virtual]
```

Get the current state of the objects connection.

## Returns:

- the current connection state

Implemented in [NetworkTable](#), [NetworkTableClient](#), and [NetworkTableServer](#).

```
virtual bool IRemote::IsServer ( ) [pure virtual]
```

If the object is acting as a server.

## Returns:

- if the object is a server

Implemented in [NetworkTable](#), [NetworkTableClient](#), and [NetworkTableServer](#).

```
virtual void IRemote::RemoveConnectionListener ( IRemoteConnectionListener *
```

Unregister a listener from connection events.

## Parameters:

- listener** the listener to be unregistered

Implemented in **NetworkTable**, and **NetworkTableNode**.

---

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

- C:/WindRiver/workspace/WPILib/tables/**IRemote.h**

Generated by  1.7.2

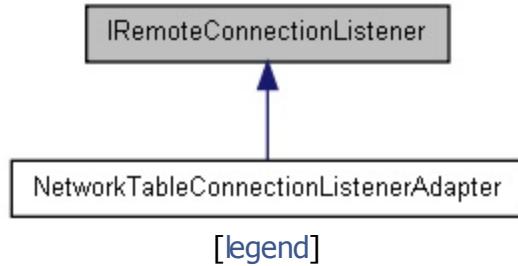


# IRemoteConnectionListener Class Reference

A listener that listens for connection changes in a **IRemote** object. More...

```
#include <IRemoteConnectionListener.h>
```

Inheritance diagram for IRemoteConnectionListener:



List of all members.

## Public Member Functions

virtual void **Connected** (**IRemote** \*remote)=0

Called when an **IRemote** is connected.

virtual void **Disconnected** (**IRemote** \*remote)=0

Called when an **IRemote** is disconnected.

---

## Detailed Description

A listener that listens for connection changes in a [IRemote](#) object.

### Author:

Mitchell

---

# Member Function Documentation

## **virtual void IRemoteConnectionListener::Connected ( IRemote \* remote )** [pure virtual]

Called when an **IRemote** is connected.

### Parameters:

**remote** the object that connected

Implemented in **NetworkTableConnectionListenerAdapter**.

## **virtual void IRemoteConnectionListener::Disconnected ( IRemote \* remote )** [pure virtual]

Called when an **IRemote** is disconnected.

### Parameters:

**remote** the object that disconnected

Implemented in **NetworkTableConnectionListenerAdapter**.

---

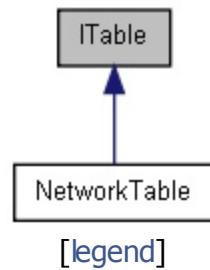
The documentation for this class was generated from the following file:

- C:/WindRiver/workspace/WPILib/tables/**IRemoteConnectionListener.h**



# ITable Class Reference

Inheritance diagram for ITable:



[List of all members.](#)

# Public Member Functions

virtual bool	<b>ContainsKey</b> (std::string key)=0
virtual bool	<b>ContainsSubTable</b> (std::string key)=0
virtual ITable *	<b>GetSubTable</b> (std::string key)=0
virtual EntryValue	<b>GetValue</b> (std::string key)=0 Gets the value associated with a key as an object.
virtual void	<b>PutValue</b> (std::string key, <b>ComplexData</b> &value)=0 Put a value in the table.
virtual void	<b>RetrieveValue</b> (std::string key, <b>ComplexData</b> &externalValue)=0
virtual void	<b>PutNumber</b> (std::string key, double value)=0 Put a number in the table.
virtual double	<b>GetNumber</b> (std::string key)=0
virtual double	<b>GetNumber</b> (std::string key, double defaultValue)=0
virtual void	<b>PutString</b> (std::string key, std::string value)=0 Put a std::string& in the table.
virtual std::string	<b>GetString</b> (std::string key)=0
virtual std::string	<b>GetString</b> (std::string key, std::string defaultValue)=0
virtual void	<b>PutBoolean</b> (std::string key, bool value)=0 Put a boolean in the table.
virtual bool	<b>GetBoolean</b> (std::string key)=0
virtual bool	<b>GetBoolean</b> (std::string key, bool defaultValue)=0
virtual void	<b>AddTableListener</b> (ITableListener *listener)=0 Add a listener for changes to the table.
virtual void	<b>AddTableListener</b> (ITableListener *listener, bool immediateNotify)=0 Add a listener for changes to the table.
virtual void	<b>AddTableListener</b> (std::string key, ITableListener *listener, bool immediateNotify)=0 Add a listener for changes to a specific key the table.
virtual void	<b>AddSubTableListener</b> (ITableListener *listener)=0 This will immediately notify the listener of all current sub tables.
virtual void	<b>RemoveTableListener</b> (ITableListener *listener)=0 Remove a listener from receiving table events.

# Member Function Documentation

**virtual void ITable::AddSubTableListener( ITableListener \* listener )** [pure virtual]

This will immediately notify the listener of all current sub tables.

## Parameters:

**listener**

Implemented in **NetworkTable**.

**virtual void ITable::AddTableListener( ITableListener \* listener )** [pure virtual]

Add a listener for changes to the table.

## Parameters:

**listener** the listener to add

Implemented in **NetworkTable**.

**virtual void ITable::AddTableListener( ITableListener \* listener,  
bool immediateNotify )** [pure virtual]

Add a listener for changes to the table.

## Parameters:

**listener** the listener to add

**immediateNotify** if true then this listener will be notified of all current entries  
(marked as new)

Implemented in **NetworkTable**.

**virtual void ITable::AddTableListener( std::string key,  
ITableListener \* listener,  
bool immediateNotify )** [pure virtual]

Add a listener for changes to a specific key the table.

#### Parameters:

<b>key</b>	the key to listen for
<b>listener</b>	the listener to add
<b>immediateNotify</b>	if true then this listener will be notified of all current entries (marked as new)

Implemented in **NetworkTable**.

### **virtual bool ITable::ContainsKey ( std::string key ) [pure virtual]**

#### Parameters:

**key** the key to search for

#### Returns:

true if the table has a value assigned to the given key

Implemented in **NetworkTable**.

### **virtual bool ITable::ContainsSubTable ( std::string key ) [pure virtual]**

#### Parameters:

**key** the key to search for

#### Returns:

true if there is a subtable with the key which contains at least one key/subtable of its own

Implemented in **NetworkTable**.

### **virtual bool ITable::GetBoolean ( std::string key ) [pure virtual]**

#### Parameters:

**key** the key to look up

#### Returns:

the value associated with the given key

## Exceptions:

**TableKeyNotDefinedException** if there is no value associated with the given key

Implemented in **NetworkTable**.

```
virtual bool ITable::GetBoolean ( std::string key,  
                                bool      defaultValue  
                            )           [pure virtual]
```

## Parameters:

**key** the key to look up

**defaultValue** the value to be returned if no value is found

## Returns:

the value associated with the given key or the given default value if there is no value associated with the key

Implemented in **NetworkTable**.

```
virtual double ITable::GetNumber ( std::string key ) [pure virtual]
```

## Parameters:

**key** the key to look up

## Returns:

the value associated with the given key

## Exceptions:

**TableKeyNotDefinedException** if there is no value associated with the given key

Implemented in **NetworkTable**.

```
virtual double ITable::GetNumber ( std::string key,  
                                 double    defaultValue  
                             )           [pure virtual]
```

## Parameters:

**key** the key to look up

**defaultValue** the value to be returned if no value is found

### Returns:

the value associated with the given key or the given default value if there is no value associated with the key

Implemented in **NetworkTable**.

**virtual std::string ITable::GetString ( std::string key ) [pure virtual]**

### Parameters:

**key** the key to look up

### Returns:

the value associated with the given key

### Exceptions:

**TableKeyNotDefinedException** if there is no value associated with the given key

Implemented in **NetworkTable**.

**virtual std::string ITable::GetString ( std::string key,  
std::string defaultValue  
) [pure virtual]**

### Parameters:

**key** the key to look up

**defaultValue** the value to be returned if no value is found

### Returns:

the value associated with the given key or the given default value if there is no value associated with the key

Implemented in **NetworkTable**.

**virtual ITable\* ITable::GetSubTable ( std::string key ) [pure virtual]**

**Parameters:**

**key** the name of the table relative to this one

**Returns:**

a sub table relative to this one

Implemented in **NetworkTable**.

**virtual EntryValue ITable::GetValue ( std::string key ) [pure virtual]**

Gets the value associated with a key as an object.

**Parameters:**

**key** the key of the value to look up

**Returns:**

the value associated with the given key

**Exceptions:**

**TableKeyNotDefinedException** if there is no value associated with the given key

Implemented in **NetworkTable**.

**virtual void ITable::PutBoolean ( std::string key,  
                                bool value  
                                ) [pure virtual]**

Put a boolean in the table.

**Parameters:**

**key** the key to be assigned to

**value** the value that will be assigned

Implemented in **NetworkTable**.

**virtual void ITable::PutNumber ( std::string key,  
                                double value  
                                ) [pure virtual]**

Put a number in the table.

#### Parameters:

**key** the key to be assigned to  
**value** the value that will be assigned

Implemented in **NetworkTable**.

```
virtual void ITable::PutString ( std::string key,  
                                std::string value  
                            ) [pure virtual]
```

Put a std::string& in the table.

#### Parameters:

**key** the key to be assigned to  
**value** the value that will be assigned

Implemented in **NetworkTable**.

```
virtual void ITable::PutValue ( std::string key,  
                               ComplexData & value  
                           ) [pure virtual]
```

Put a value in the table.

#### Parameters:

**key** the key to be assigned to  
**value** the value that will be assigned

#### Exceptions:

**IllegalArgumentException** when the value is not supported by the table

Implemented in **NetworkTable**.

```
virtual void ITable::RemoveTableListener ( ITableListener * listener ) [pure vi
```

Remove a listener from receiving table events.

## Parameters:

**listener** the listener to be removed

Implemented in **NetworkTable**.

---

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

- C:/WindRiver/workspace/WPILib/tables/**ITable.h**

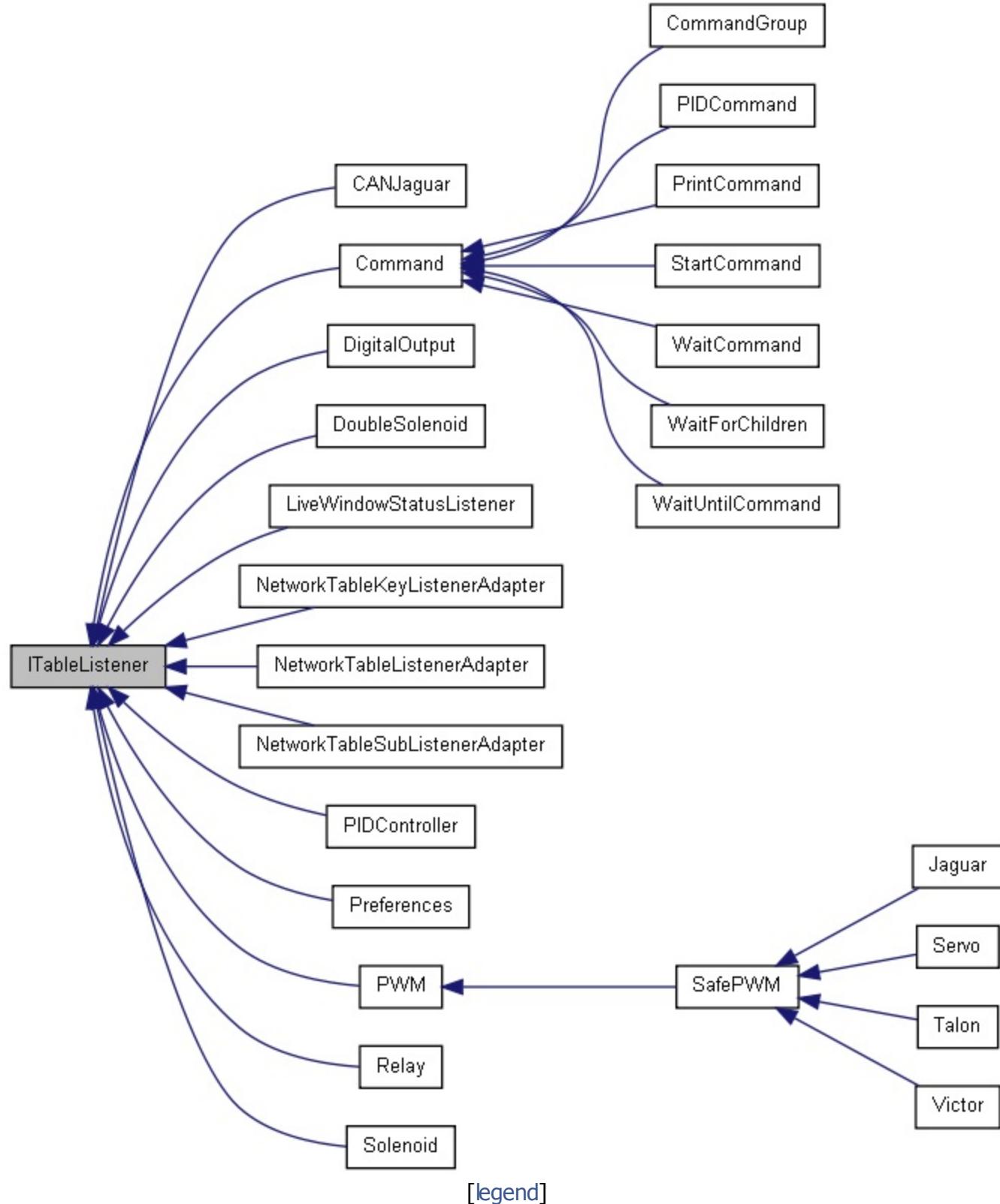


# ITableListener Class Reference

A listener that listens to changes in values in a **ITable**. More...

```
#include <ITableListener.h>
```

Inheritance diagram for ITableListener:



List of all members.

virtual void **ValueChanged** (**ITable** \*source, const std::string &key, **EntryValue** value, bool isNew)=0  
Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

---

## Detailed Description

A listener that listens to changes in values in a [ITable](#).

### Author:

Mitchell

---

# Member Function Documentation

```
virtual void ITableListener::ValueChanged( ITable * source,
                                         const std::string & key,
                                         EntryValue value,
                                         bool isNew
                                         )
                                         [pure virtual]
```

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

## Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implemented in **CANJaguar**, **Command**, **DigitalOutput**, **DoubleSolenoid**, **LiveWindowStatusListener**, **NetworkTableKeyListenerAdapter**, **NetworkTableListenerAdapter**, **NetworkTableSubListenerAdapter**, **Preferences**, **PWM**, **Relay**, **Servo**, and **Solenoid**.

---

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

- C:/WindRiver/workspace/WPIlib/tables/**ITableListener.h**

[Class List](#)[Class Hierarchy](#)[Class Members](#)

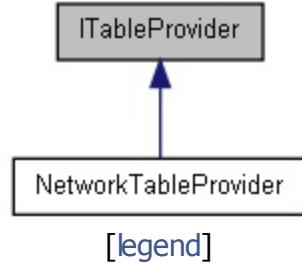
# ITableProvider Class Reference

---

A simple interface to provide tables. More...

```
#include <ITableProvider.h>
```

Inheritance diagram for ITableProvider:



---

List of all members.

# Detailed Description

A simple interface to provide tables.

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/tables/**ITableProvider.h**

---

Generated by  1.7.2

[Class List](#)[Class Hierarchy](#)[Class Members](#)

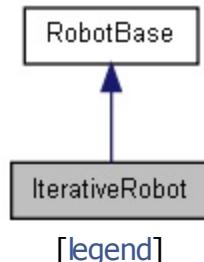
[Public Member Functions](#) |  
[Static Public Attributes](#) |  
[Protected Member Functions](#)

# IterativeRobot Class Reference

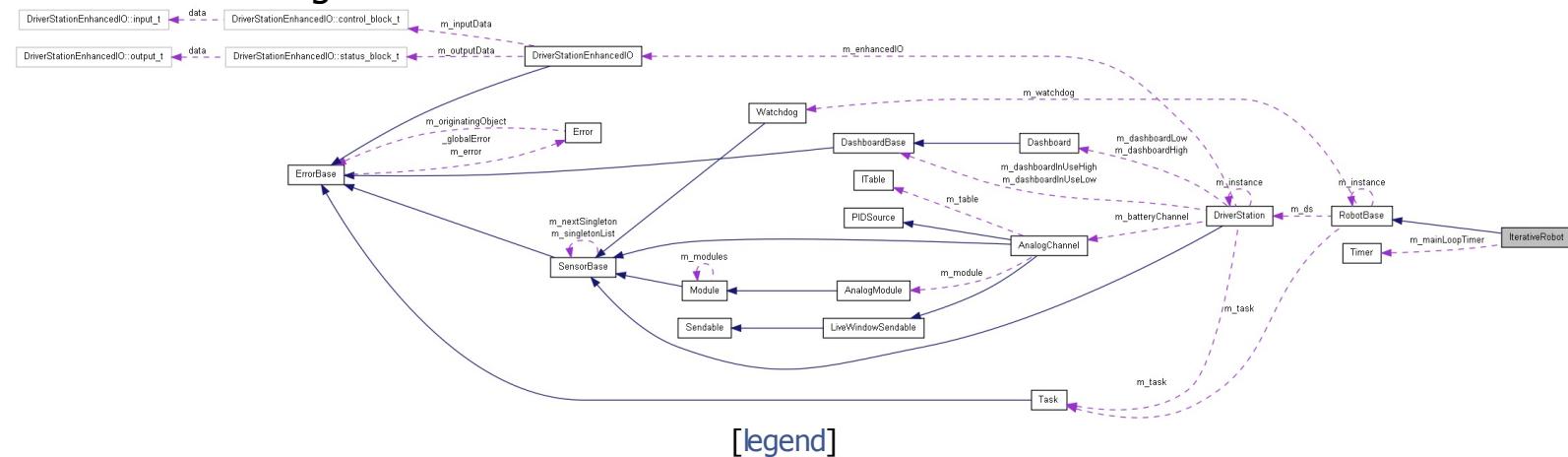
**IterativeRobot** implements a specific type of Robot Program framework, extending the **RobotBase** class. [More...](#)

```
#include <IterativeRobot.h>
```

## Inheritance diagram for IterativeRobot:



## Collaboration diagram for IterativeRobot:



## List of all members.

virtual void	<b>StartCompetition ()</b>	Provide an alternate "main loop" via <b>StartCompetition()</b> .
virtual void	<b>RobotInit ()</b>	Robot-wide initialization code should go here.
virtual void	<b>DisabledInit ()</b>	Initialization code for disabled mode should go here.
virtual void	<b>AutonomousInit ()</b>	Initialization code for autonomous mode should go here.
virtual void	<b>TeleopInit ()</b>	Initialization code for teleop mode should go here.
virtual void	<b>TestInit ()</b>	Initialization code for test mode should go here.

virtual void **DisabledPeriodic** ()  
Periodic code for disabled mode should go here.

virtual void **AutonomousPeriodic** ()  
Periodic code for autonomous mode should go here.

virtual void **TeleopPeriodic** ()  
Periodic code for teleop mode should go here.

virtual void **TestPeriodic** ()  
Periodic code for test mode should go here.

void **SetPeriod** (double period)  
Set the period for the periodic functions.

double **GetPeriod** ()  
Get the period for the periodic functions.

double **GetLoopsPerSec** ()  
Get the number of loops per second for the **IterativeRobot**.

static const double **kDefaultPeriod** = 0.0

virtual **~IterativeRobot** ()  
Free the resources for a RobotIterativeBase class.

**IterativeRobot** ()



## Detailed Description

**IterativeRobot** implements a specific type of Robot Program framework, extending the **RobotBase** class.

The **IterativeRobot** class is intended to be subclassed by a user creating a robot program.

This class is intended to implement the "old style" default code, by providing the following functions which are called by the main loop, **StartCompetition()**, at the appropriate times:

**RobotInit()** -- provide for initialization at robot power-on

Init() functions -- each of the following functions is called once when the appropriate mode is entered:

- **DisabledInit()** -- called only when first disabled
- **AutonomousInit()** -- called each and every time autonomous is entered from another mode
- **TeleopInit()** -- called each and every time teleop is entered from another mode
- **TestInit()** -- called each and every time test is entered from another mode

Periodic() functions -- each of these functions is called iteratively at the appropriate periodic rate (aka the "slow loop"). The default period of the iterative robot is synced to the driver station control packets, giving a periodic frequency of about 50Hz (50 times per second).

- **DisabledPeriodic()**
  - **AutonomousPeriodic()**
  - **TeleopPeriodic()**
  - **TestPeriodic()**
-

# Constructor & Destructor Documentation

## **IterativeRobot::IterativeRobot( )** [protected]

Constructor for RobotIterativeBase.

The constructor initializes the instance variables for the robot to indicate the status of initialization for disabled, autonomous, teleop, and test code.

# Member Function Documentation

## **void IterativeRobot::AutonomousInit( ) [virtual]**

Initialization code for autonomous mode should go here.

Users should override this method for initialization code which will be called each time the robot enters autonomous mode.

## **void IterativeRobot::AutonomousPeriodic( ) [virtual]**

Periodic code for autonomous mode should go here.

Users should override this method for code which will be called periodically at a regular rate while the robot is in autonomous mode.

## **void IterativeRobot::DisabledInit( ) [virtual]**

Initialization code for disabled mode should go here.

Users should override this method for initialization code which will be called each time the robot enters disabled mode.

## **void IterativeRobot::DisabledPeriodic( ) [virtual]**

Periodic code for disabled mode should go here.

Users should override this method for code which will be called periodically at a regular rate while the robot is in disabled mode.

## **double IterativeRobot::GetLoopsPerSec( )**

Get the number of loops per second for the **IterativeRobot**.

### **Returns:**

Frequency of the periodic function calls

## **double IterativeRobot::GetPeriod( )**

Get the period for the periodic functions.

Returns 0.0 if configured to syncronize with DS control data packets.

### Returns:

Period of the periodic function calls

## **void IterativeRobot::RobotInit( ) [virtual]**

Robot-wide initialization code should go here.

Users should override this method for default Robot-wide initialization which will be called when the robot is first powered on. It will be called exactly 1 time.

## **void IterativeRobot::SetPeriod ( double period )**

Set the period for the periodic functions.

### Parameters:

**period** The period of the periodic function calls. 0.0 means sync to driver station control data.

## **void IterativeRobot::StartCompetition( ) [virtual]**

Provide an alternate "main loop" via **StartCompetition()**.

This specific **StartCompetition()** implements "main loop" behavior like that of the FRC control system in 2008 and earlier, with a primary (slow) loop that is called periodically, and a "fast loop" (a.k.a. "spin loop") that is called as fast as possible with no delay between calls.

Implements **RobotBase**.

## **void IterativeRobot::TeleopInit( ) [virtual]**

Initialization code for teleop mode should go here.

Users should override this method for initialization code which will be called each time the robot enters teleop mode.

## **void IterativeRobot::TeleopPeriodic( )** [virtual]

Periodic code for teleop mode should go here.

Users should override this method for code which will be called periodically at a regular rate while the robot is in teleop mode.

## **void IterativeRobot::TestInit( )** [virtual]

Initialization code for test mode should go here.

Users should override this method for initialization code which will be called each time the robot enters test mode.

## **void IterativeRobot::TestPeriodic( )** [virtual]

Periodic code for test mode should go here.

Users should override this method for code which will be called periodically at a regular rate while the robot is in test mode.

---

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

- C:/WindRiver/workspace/WPILib/**IterativeRobot.h**
- C:/WindRiver/workspace/WPILib/IterativeRobot.cpp

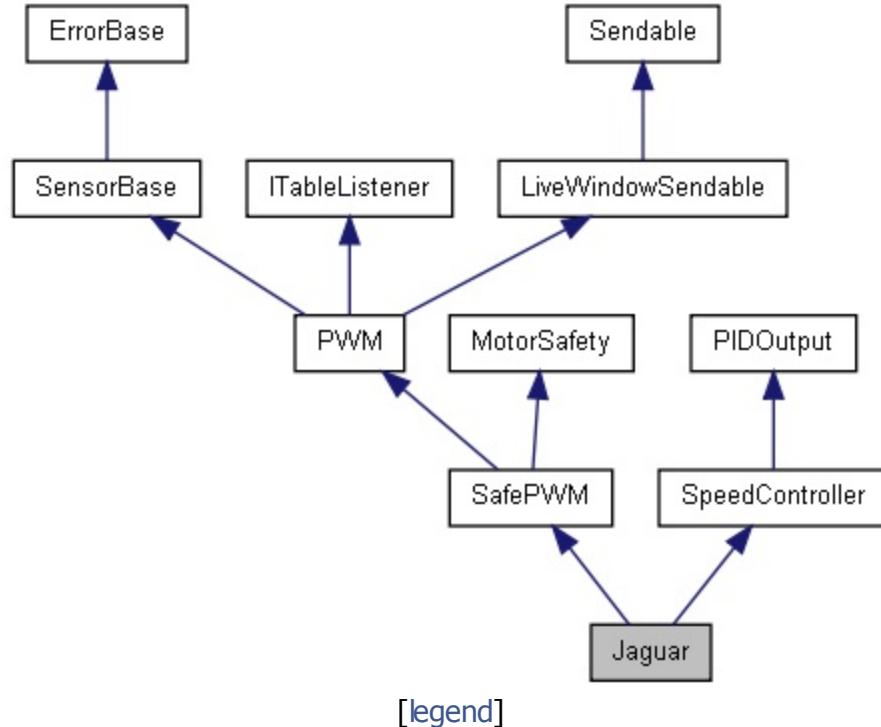


# Jaguar Class Reference

Luminary Micro **Jaguar** Speed Control. More...

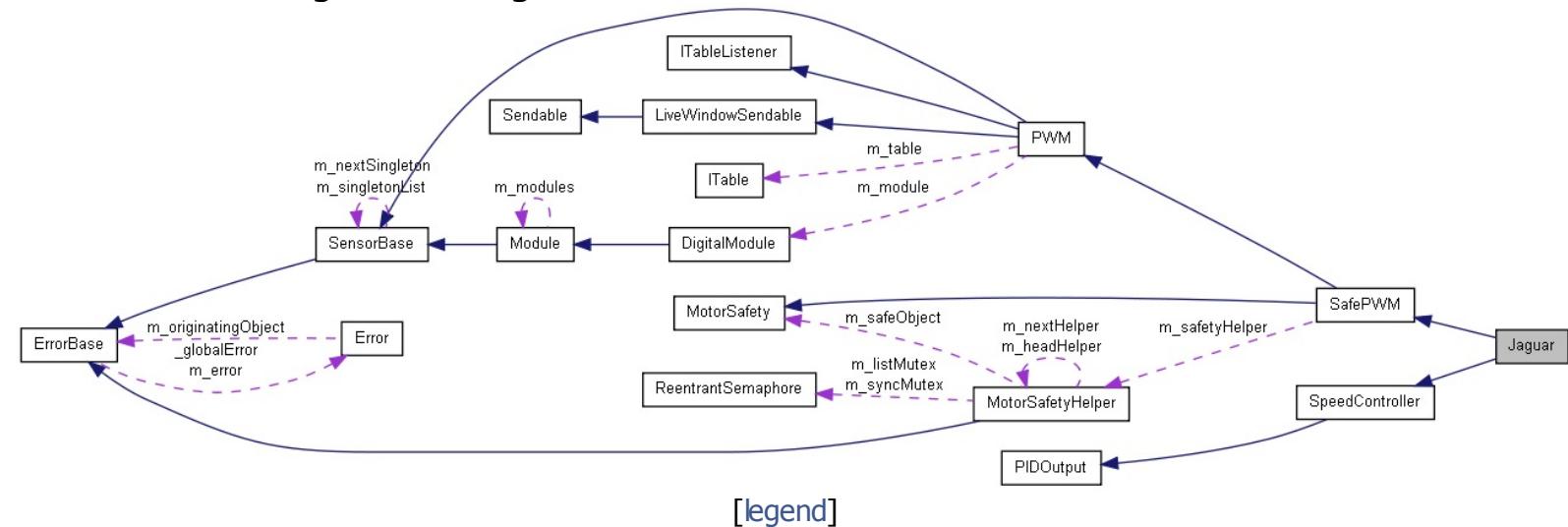
```
#include <Jaguar.h>
```

Inheritance diagram for Jaguar:



[legend]

Collaboration diagram for Jaguar:



[legend]

List of all members.

# Public Member Functions

**Jaguar** (UINT32 channel)

Constructor that assumes the default digital module.

**Jaguar** (UINT8 moduleNumber, UINT32 channel)

Constructor that specifies the digital module.

virtual void **Set** (float value, UINT8 syncGroup=0)

Set the **PWM** value.

virtual float **Get** ()

Get the recently set value of the **PWM**.

virtual void **Disable** ()

Common interface for disabling a motor.

virtual void **PIDWrite** (float output)

Write out the PID value as seen in the **PIDOutput** base object.

---

## Detailed Description

Luminary Micro **Jaguar** Speed Control.

---

# Constructor & Destructor Documentation

## Jaguar::Jaguar ( **UINT32 channel** ) [explicit]

Constructor that assumes the default digital module.

### Parameters:

**channel** The **PWM** channel on the digital module that the **Jaguar** is attached to.

## Jaguar::Jaguar ( **UINT8 moduleNumber,** **UINT32 channel** )

Constructor that specifies the digital module.

### Parameters:

**moduleNumber** The digital module (1 or 2).

**channel** The **PWM** channel on the digital module that the **Jaguar** is attached to.

# Member Function Documentation

## **float Jaguar::Get( ) [virtual]**

Get the recently set value of the **PWM**.

### Returns:

The most recently set value for the **PWM** between -1.0 and 1.0.

Implements **SpeedController**.

## **void Jaguar::PIDWrite( float output ) [virtual]**

Write out the PID value as seen in the **PIDOutput** base object.

### Parameters:

**output** Write out the **PWM** value as was found in the **PIDController**

Implements **PIDOutput**.

## **void Jaguar::Set( float speed,                   UINT8 syncGroup = 0                   ) [virtual]**

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:

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

**syncGroup** Unused interface.

Implements **SpeedController**.

---

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

- C:/WindRiver/workspace/WPILib/**Jaguar.h**
- C:/WindRiver/workspace/WPILib/Jaguar.cpp



[Class List](#)[Class Hierarchy](#)[Class Members](#)[Skeleton](#)[Joint](#)

Public Attributes

# Skeleton::Joint Struct Reference

---

List of all members.

## Public Attributes

float **x**

float **y**

float **z**

JointTrackingState **trackingState**

---

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

- C:/WindRiver/workspace/WPILib/**Skeleton.h**

[Class List](#)[Class Hierarchy](#)[Class Members](#)

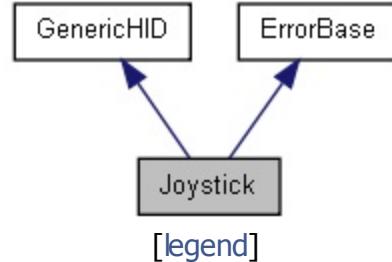
[Public Types](#) | [Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Static Public Attributes](#)

# Joystick Class Reference

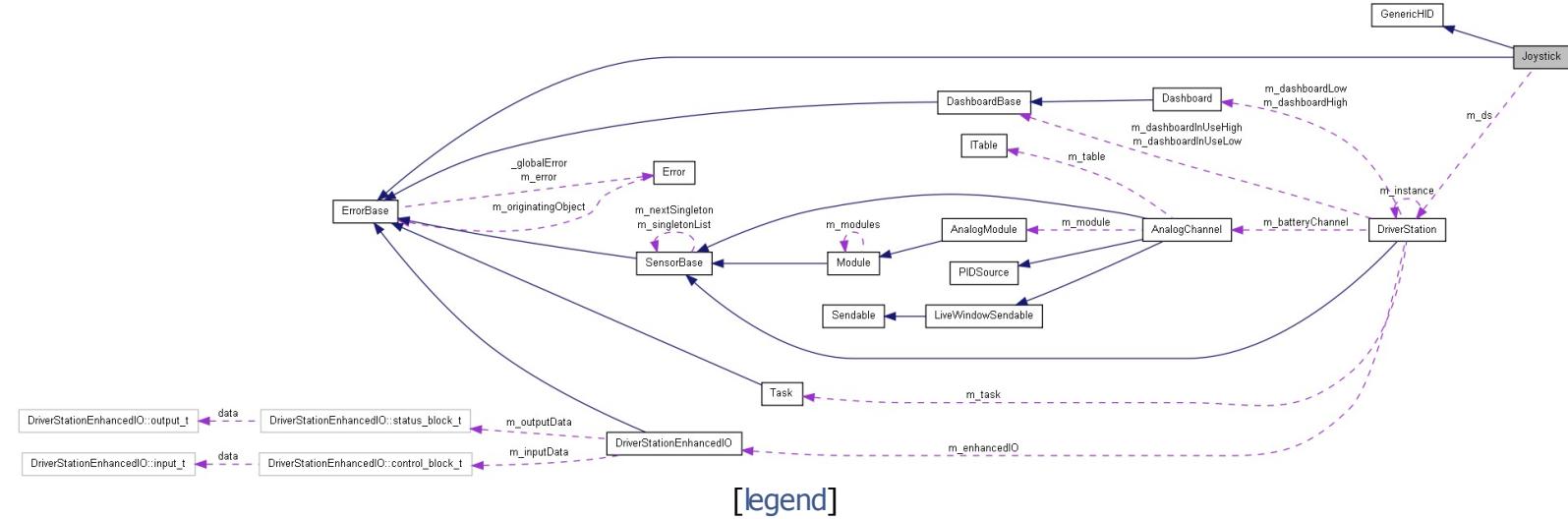
Handle input from standard Joysticks connected to the Driver Station. [More...](#)

```
#include <Joystick.h>
```

Inheritance diagram for Joystick:



Collaboration diagram for Joystick:



List of all members.

enum	<b>AxisType {</b> <b>kXAxis, kYAxis, kZAxis, kTwistAxis,</b> <b>kThrottleAxis, kNumAxisTypes</b> <b>}</b>
enum	<b>ButtonType {</b> <b>kTriggerButton, kTopButton,</b> <b>kNumButtonTypes</b> <b>}</b>

# Public Member Functions

**Joystick** (UINT32 port)

Construct an instance of a joystick.

**Joystick** (UINT32 port, UINT32 numAxisTypes, UINT32 numButtonTypes)

Version of the constructor to be called by sub-classes.

UINT32 **GetAxisChannel** (AxisType axis)

Get the channel currently associated with the specified axis.

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

Set the channel associated with a specified axis.

virtual float **GetX** (JoystickHand hand=kRightHand)

virtual float	<b>GetY</b> (JoystickHand hand=kRightHand) Get the Y value of the joystick.
virtual float	<b>GetZ</b> () Get the Z value of the current joystick.
virtual float	<b>GetTwist</b> () Get the twist value of the current joystick.
virtual float	<b>GetThrottle</b> () Get the throttle value of the current joystick.
virtual float	<b>GetAxis</b> (AxisType axis) For the current joystick, return the axis determined by the argument.
float	<b>GetRawAxis</b> (UINT32 axis) Get the value of the axis.
virtual bool	<b>GetTrigger</b> (JoystickHand hand=kRightHand) Read the state of the trigger on the joystick.
virtual bool	<b>GetTop</b> (JoystickHand hand=kRightHand) Read the state of the top button on the joystick.
virtual bool	<b>GetBumper</b> (JoystickHand hand=kRightHand) This is not supported for the <b>Joystick</b> .
virtual bool	<b>GetButton</b> (ButtonType button) Get buttons based on an enumerated type.
bool	<b>GetRawButton</b> (UINT32 button) Get the button value for buttons 1 through 12.
virtual float	<b>GetMagnitude</b> () Get the magnitude of the direction vector formed by the joystick's current position relative to its origin.
virtual float	<b>GetDirectionRadians</b> () Get the direction of the vector formed by the joystick and its origin in radians.
virtual float	<b>GetDirectionDegrees</b> () Get the direction of the vector formed by the joystick and its origin in degrees.

# Static Public Member Functions

---

```
static Joystick * GetStickForPort (UINT32 port)
```

## Static Public Attributes

```
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
```

## Detailed Description

Handle input from standard Joysticks connected to the Driver Station.

This class handles standard input that comes from the Driver Station. Each time a value is requested the most recent value is returned. There is a single class instance for each joystick and the mapping of ports to hardware buttons depends on the code in the driver station.

---

# Constructor & Destructor Documentation

## Joystick::Joystick ( **UINT32 port** ) [explicit]

Construct an instance of a joystick.

The joystick index is the usb port on the drivers station.

### Parameters:

**port** The port on the driver station that the joystick is plugged into.

## Joystick::Joystick ( **UINT32 port,** **UINT32 numAxisTypes,** **UINT32 numButtonTypes** )

Version of the constructor to be called by sub-classes.

This constructor allows the subclass to configure the number of constants for axes and buttons.

### Parameters:

**port** The port on the driver station that the joystick is plugged into.

**numAxisTypes** The number of axis types in the enum.

**numButtonTypes** The number of button types in the enum.

# Member Function Documentation

## **float Joystick::GetAxis ( AxisType axis ) [virtual]**

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

This is for cases where the joystick axis is returned programmaticaly, otherwise one of the previous functions would be preferable (for example [GetX\(\)](#)).

### **Parameters:**

**axis** The axis to read.

### **Returns:**

The value of the axis.

## **UINT32 Joystick::GetAxisChannel ( AxisType axis )**

Get the channel currently associated with the specified axis.

### **Parameters:**

**axis** The axis to look up the channel for.

### **Returns:**

The channel fr the axis.

## **bool Joystick::GetBumper ( JoystickHand hand = kRightHand ) [virtual]**

This is not supported for the [Joystick](#).

This method is only here to complete the [GenericHID](#) interface.

Implements [GenericHID](#).

## **bool Joystick::GetButton ( ButtonType button ) [virtual]**

Get buttons based on an enumerated type.

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

### **Parameters:**

**button** The type of button to read.

**Returns:**

The state of the button.

**float Joystick::GetDirectionDegrees( ) [virtual]**

Get the direction of the vector formed by the joystick and its origin in degrees.

uses  $\text{acos}(-1)$  to represent Pi due to absence of readily accessible Pi constant in C++

**Returns:**

The direction of the vector in degrees

**float Joystick::GetDirectionRadians( ) [virtual]**

Get the direction of the vector formed by the joystick and its origin in radians.

**Returns:**

The direction of the vector in radians

**float Joystick::GetMagnitude( ) [virtual]**

Get the magnitude of the direction vector formed by the joystick's current position relative to its origin.

**Returns:**

The magnitude of the direction vector

**float Joystick::GetRawAxis( **UINT32** axis ) [virtual]**

Get the value of the axis.

**Parameters:**

**axis** The axis to read [1-6].

**Returns:**

The value of the axis.

Implements **GenericHID**.

## **bool Joystick::GetRawButton ( **UINT32 button** ) [virtual]**

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:**

**button** The button number to be read.

### **Returns:**

The state of the button.

Implements **Generic HID**.

## **float Joystick::GetThrottle ( ) [virtual]**

Get the throttle value of the current joystick.

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

Implements **Generic HID**.

## **bool Joystick::GetTop ( **JoystickHand hand = kRightHand** ) [virtual]**

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:**

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

### **Returns:**

The state of the top button.

Implements **Generic HID**.

## **bool Joystick::GetTrigger ( **JoystickHand hand = kRightHand** ) [virtual]**

Read the state of the trigger on the joystick.

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

### Parameters:

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

### Returns:

The state of the trigger.

Implements **GenericHID**.

### **float Joystick::GetTwist( ) [virtual]**

Get the twist value of the current joystick.

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

Implements **GenericHID**.

### **float Joystick::GetX ( JoystickHand hand = kRightHand ) [virtual]**

Get the X value of the joystick.

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

Implements **GenericHID**.

### **float Joystick::GetY ( JoystickHand hand = kRightHand ) [virtual]**

Get the Y value of the joystick.

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

Implements **GenericHID**.

### **float Joystick::GetZ( ) [virtual]**

Get the Z value of the current joystick.

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

Implements [GenericHID](#).

```
void Joystick::SetAxisChannel( AxisType axis,  
                               UINT32 channel  
                           )
```

Set the channel associated with a specified axis.

#### Parameters:

**axis** The axis to set the channel for.

**channel** The channel to set the axis to.

---

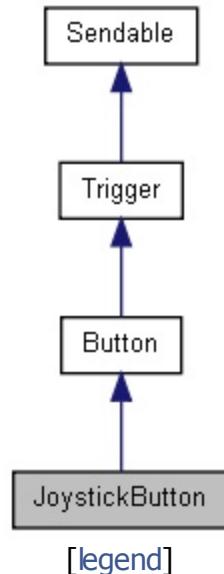
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/[Joystick.h](#)
- C:/WindRiver/workspace/WPILib/Joystick.cpp

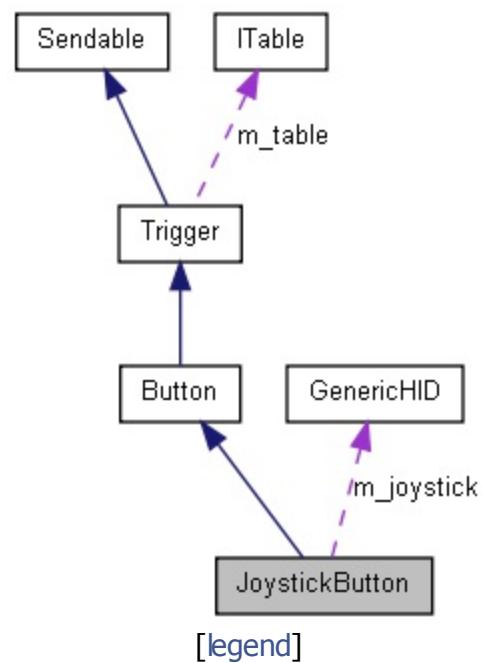


# JoystickButton Class Reference

Inheritance diagram for JoystickButton:



Collaboration diagram for JoystickButton:



List of all members.

# Public Member Functions

**JoystickButton** ([GenericHID](#) \*joystick, int buttonNumber)

virtual bool **Get** ()

---

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

- C:/WindRiver/workspace/WPILib/Buttons/[JoystickButton.h](#)
- C:/WindRiver/workspace/WPILib/Buttons/JoystickButton.cpp

---

Generated by [doxygen](#) 1.7.2

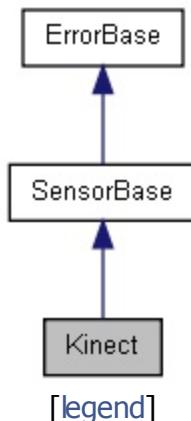
[Classes](#) | [Public Types](#) |  
[Public Member Functions](#) |  
[Static Public Member Functions](#)

# Kinect Class Reference

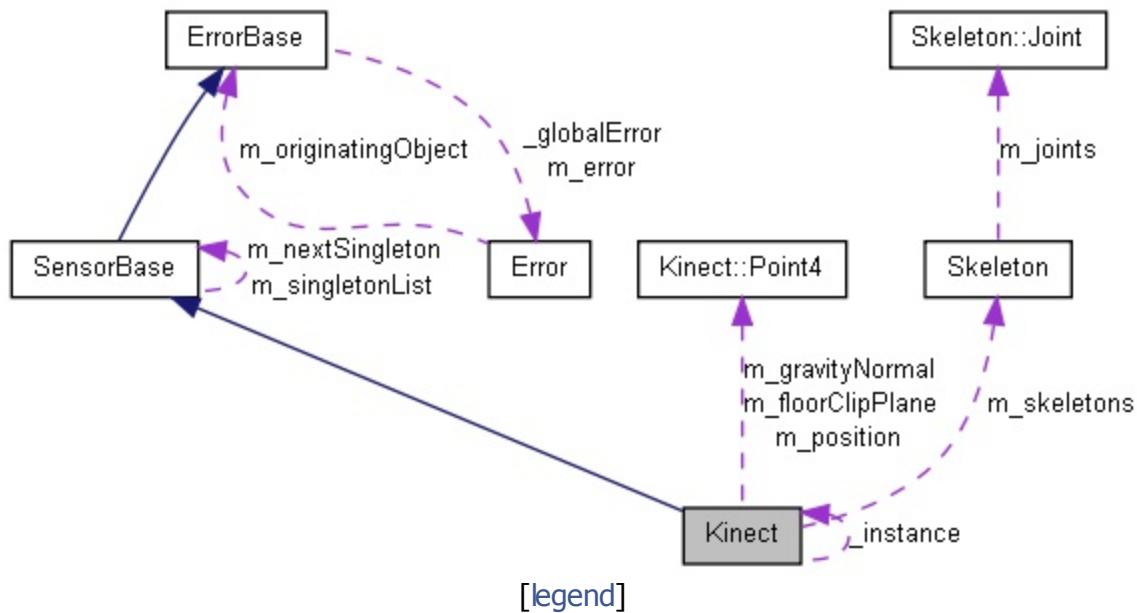
Handles raw data input from the FRC **Kinect** Server when used with a **Kinect** device connected to the Driver Station. [More...](#)

```
#include <Kinect.h>
```

Inheritance diagram for Kinect:



Collaboration diagram for Kinect:



List of all members.

# Classes

struct **Point4**

---

enum **SkeletonTrackingState** { **kNotTracked**, **kPositionOnly**,  
**kTracked** }

---

enum **SkeletonQuality** { **kClippedRight** = 1, **kClippedLeft** = 2,  
**kClippedTop** = 4, **kClippedBottom** = 8 }

# Public Member Functions

**int GetNumberOfPlayers ()**

Get the number of tracked players on the **Kinect**.

**Point4 GetFloorClipPlane ()**

Get the floor clip plane as defined in the **Kinect** SDK.

**Point4 GetGravityNormal ()**

Get the gravity normal from the kinect as defined in the **Kinect** SDK.

**Skeleton GetSkeleton (int skeletonIndex=1)**

Get the skeleton data Returns the detected skeleton data from the kinect as defined in the **Kinect** SDK.

**Point4 GetPosition (int skeletonIndex=1)**

Get the current position of the skeleton.

**UINT32 GetQuality (int skeletonIndex=1)**

Get the quality of the skeleton.

**SkeletonTrackingState GetTrackingState (int skeletonIndex=1)**

Get the TrackingState of the skeleton.

# Static Public Member Functions

**static Kinect \* GetInstance ()**

Get the one and only **Kinect** object.

---

## Detailed Description

Handles raw data input from the FRC **Kinect** Server when used with a **Kinect** device connected to the Driver Station.

Each time a value is requested the most recent value is returned. See Getting Started with Microsoft **Kinect** for FRC and the **Kinect** for Windows SDK API reference for more information

---

# Member Function Documentation

## **Kinect::Point4 Kinect::GetFloorClipPlane( )**

Get the floor clip plane as defined in the **Kinect** SDK.

### Returns:

The floor clip plane

## **Kinect::Point4 Kinect::GetGravityNormal( )**

Get the gravity normal from the kinect as defined in the **Kinect** SDK.

### Returns:

The gravity normal (w is ignored)

## **Kinect \* Kinect::GetInstance( ) [static]**

Get the one and only **Kinect** object.

### Returns:

pointer to a **Kinect**

## **int Kinect::GetNumberOfPlayers( )**

Get the number of tracked players on the **Kinect**.

### Returns:

the number of players being actively tracked

## **Kinect::Point4 Kinect::GetPosition( int skeletonIndex = 1 )**

Get the current position of the skeleton.

### Parameters:

**skeletonIndex** the skeleton to read from

### Returns:

the current position as defined in the **Kinect** SDK (w is ignored)

## **UINT32 Kinect::GetQuality ( int skeletonIndex = 1 )**

Get the quality of the skeleton.

Quality masks are defined in the SkeletonQuality enum

### **Parameters:**

**skeletonIndex** the skeleton to read from

### **Returns:**

the quality value as defined in the **Kinect** SDK

## **Skeleton Kinect::GetSkeleton ( int skeletonIndex = 1 )**

Get the skeleton data Returns the detected skeleton data from the kinect as defined in the **Kinect** SDK.

### **Parameters:**

**skeletonIndex** Which of (potentially 2) skeletons to return. This is ignored in this implementation and only a single skeleton is supported for the FRC release default gesture interpretation.

### **Returns:**

The current version of the skeleton object.

## **Kinect::SkeletonTrackingState Kinect::GetTrackingState ( int skeletonIndex = 1 )**

Get the TrackingState of the skeleton.

Tracking states are defined in the SkeletonTrackingState enum

### **Parameters:**

**skeletonIndex** the skeleton to read from

### **Returns:**

the tracking state value as defined in the **Kinect** SDK

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

- C:/WindRiver/workspace/WPILib/**Kinect.h**
  - C:/WindRiver/workspace/WPILib/Kinect.cpp
- 

Generated by [doxygen](#) 1.7.2

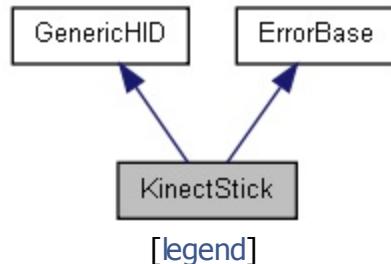


# KinectStick Class Reference

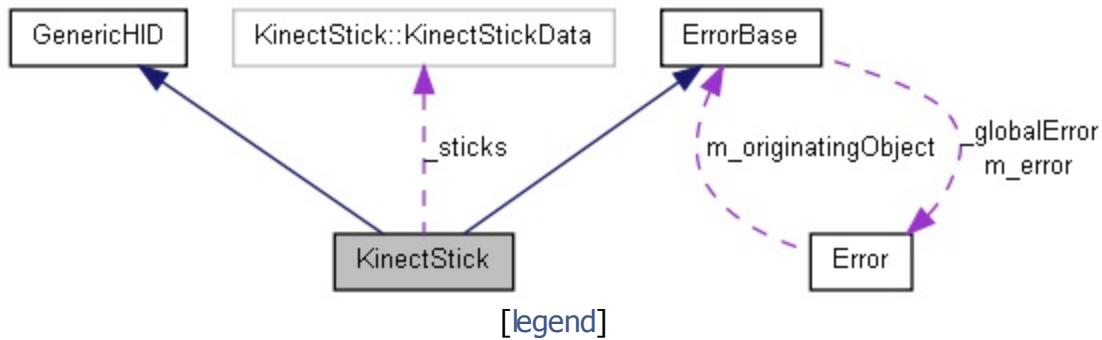
Handles input from the **Joystick** data sent by the FRC **Kinect** Server when used with a **Kinect** device connected to the Driver Station. [More...](#)

```
#include <KinectStick.h>
```

Inheritance diagram for KinectStick:



Collaboration diagram for KinectStick:



List of all members.

# Classes

union **KinectStickData**

**KinectStick** (int id)

**Kinect** joystick constructor.

virtual float **GetX** (JoystickHand hand=kRightHand)  
Get the X value of the **KinectStick**.

virtual float **GetY** (JoystickHand hand=kRightHand)  
Get the Y value of the **KinectStick**.

virtual float **GetZ** ()  
Get the Z value of the **KinectStick**.

virtual float **GetTwist** ()  
Get the Twist value of the **KinectStick**.

virtual float **GetThrottle** ()  
Get the Throttle value of the **KinectStick**.

virtual float **GetRawAxis** (UINT32 axis)  
Get the value of the **KinectStick** axis.

virtual bool **GetTrigger** (JoystickHand hand=kRightHand)  
Get the button value for the button set as the default trigger.

virtual bool **GetTop** (JoystickHand hand=kRightHand)  
Get the button value for the button set as the default top.

virtual bool **GetBumper** (JoystickHand hand=kRightHand)  
Get the button value for the button set as the default bumper (button 4)

virtual bool **GetRawButton** (UINT32 button)

Get the button value for buttons 1 through 12.

---

## Detailed Description

Handles input from the **Joystick** data sent by the FRC **Kinect** Server when used with a **Kinect** device connected to the Driver Station.

Each time a value is requested the most recent value is returned. Default gestures embedded in the FRC **Kinect** Server are described in the document Getting Started with Microsoft **Kinect** for FRC.

---

# Constructor & Destructor Documentation

## KinectStick::KinectStick( int id ) [explicit]

**Kinect** joystick constructor.

### Parameters:

**id** value is either 1 or 2 for the left or right joystick decoded from gestures interpreted by the **Kinect** server on the Driver Station computer.

# Member Function Documentation

**bool KinectStick::GetBumper ( JoystickHand hand = kRightHand ) [virtual]**

Get the button value for the button set as the default bumper (button 4)

**Parameters:**

**hand** Unused

**Returns:**

The state of the button.

Implements **Generic HID**.

**float KinectStick::GetRawAxis ( UINT32 axis ) [virtual]**

Get the value of the **KinectStick** axis.

**Parameters:**

**axis** The axis to read [1-6].

**Returns:**

The value of the axis

Implements **Generic HID**.

**bool KinectStick::GetRawButton ( UINT32 button ) [virtual]**

Get the button value for buttons 1 through 12.

The default gestures implement only 9 buttons.

The appropriate button is returned as a boolean value.

**Parameters:**

**button** The button number to be read.

**Returns:**

The state of the button.

Implements **Generic HID**.

**float KinectStick::GetThrottle( ) [virtual]**

Get the Throttle value of the **KinectStick**.

This axis is unimplemented in the default gestures but can be populated by teams editing the **Kinect** Server.

**Returns:**

The Throttle value of the **KinectStick**

Implements **GenericID**.

**bool KinectStick::GetTop ( JoystickHand hand = kRightHand ) [virtual]**

Get the button value for the button set as the default top.

**Parameters:**

**hand** Unused

**Returns:**

The state of the button.

Implements **GenericID**.

**bool KinectStick::GetTrigger ( JoystickHand hand = kRightHand ) [virtual]**

Get the button value for the button set as the default trigger.

**Parameters:**

**hand** Unused

**Returns:**

The state of the button.

Implements **GenericID**.

**float KinectStick::GetTwist( ) [virtual]**

Get the Twist value of the **KinectStick**.

This axis is unimplemented in the default gestures but can be populated by teams editing the **Kinect** Server.

### Returns:

The Twist value of the **KinectStick**

Implements **GenericID**.

**float KinectStick::GetX ( JoystickHand hand = kRightHand ) [virtual]**

Get the X value of the **KinectStick**.

This axis is unimplemented in the default gestures but can be populated by teams editing the **Kinect** Server.

### Parameters:

**hand** Unused

### Returns:

The X value of the **KinectStick**

Implements **GenericID**.

**float KinectStick::GetY ( JoystickHand hand = kRightHand ) [virtual]**

Get the Y value of the **KinectStick**.

This axis represents arm angle in the default gestures

### Parameters:

**hand** Unused

### Returns:

The Y value of the **KinectStick**

Implements **GenericID**.

**float KinectStick::GetZ ( ) [virtual]**

Get the Z value of the **KinectStick**.

This axis is unimplemented in the default gestures but can be populated by teams editing the **Kinect** Server.

**Parameters:**

**hand** Unused

**Returns:**

The Z value of the **KinectStick**

Implements **GenericHID**.

---

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

- C:/WindRiver/workspace/WPILib/**KinectStick.h**
- C:/WindRiver/workspace/WPILib/KinectStick.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

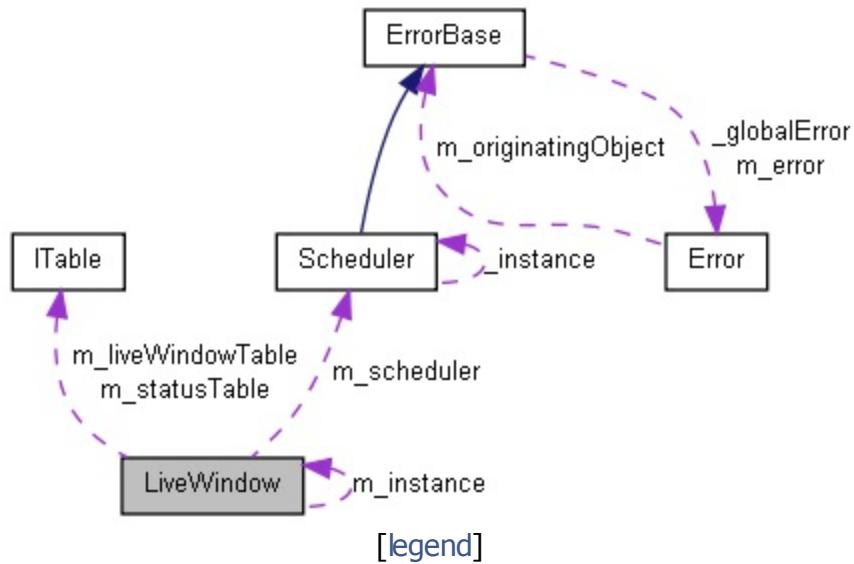
[Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Protected Member Functions](#)

# LiveWindow Class Reference

The **LiveWindow** class is the public interface for putting sensors and actuators on the **LiveWindow**. More...

```
#include <LiveWindow.h>
```

Collaboration diagram for LiveWindow:



List of all members.

# Public Member Functions

void **Run ()**

This method is called periodically to cause the sensors to send new values to the **SmartDashboard**.

void **AddSensor (char \*subsystem, char \*name,  
LiveWindowSendable \*component)**

Add a Sensor associated with the subsystem and with call it by the given name.

void **AddActuator (char \*subsystem, char \*name,  
LiveWindowSendable \*component)**

Add an Actuator associated with the subsystem and with call it by the given name.

void **AddComponent (char \*subsystem, char \*name,  
LiveWindowSendable \*component)**

Add a Component associated with the subsystem and with call it by the given name.

bool **IsEnabled ()**

void **SetEnabled (bool enabled)**

Change the enabled status of **LiveWindow** If it changes to enabled, start livewindow running otherwise stop it.

# Static Public Member Functions

---

static **LiveWindow** \* **GetInstance** ()

Get an instance of the **LiveWindow** main class This is a singleton to guarantee that there is only a single instance regardless of how many times GetInstance is called.

# Protected Member Functions

---

**LiveWindow ()**  
**LiveWindow** constructor.

---

## Detailed Description

The **LiveWindow** class is the public interface for putting sensors and actuators on the **LiveWindow**.

### Author:

Brad Miller

---

# Constructor & Destructor Documentation

## **LiveWindow::LiveWindow( )** [protected]

**LiveWindow** constructor.

Allocate the necessary tables.

# Member Function Documentation

```
void LiveWindow::AddActuator( char * subsystem,  
                               char * name,  
                               LiveWindowSendable * component  
                           )
```

Add an Actuator associated with the subsystem and with call it by the given name.

## Parameters:

**subsystem** The subsystem this component is part of.  
**name** The name of this component.  
**component** A **LiveWindowSendable** component that represents a actuator.

```
void LiveWindow::AddComponent( char * subsystem,  
                               char * name,  
                               LiveWindowSendable * component  
                           )
```

Add a Component associated with the subsystem and with call it by the given name.

## Parameters:

**subsystem** The subsystem this component is part of.  
**name** The name of this component.  
**component** A **LiveWindowSendable** component that represents a component.

```
void LiveWindow::AddSensor( char * subsystem,  
                           char * name,  
                           LiveWindowSendable * component  
                         )
```

Add a Sensor associated with the subsystem and with call it by the given name.

## Parameters:

**subsystem** The subsystem this component is part of.  
**name** The name of this component.  
**component** A **LiveWindowSendable** component that represents a sensor.

---

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

- C:/WindRiver/workspace/WPILib/LiveWindow/**LiveWindow.h**
- C:/WindRiver/workspace/WPILib/LiveWindow/LiveWindow.cpp

---

Generated by  1.7.2

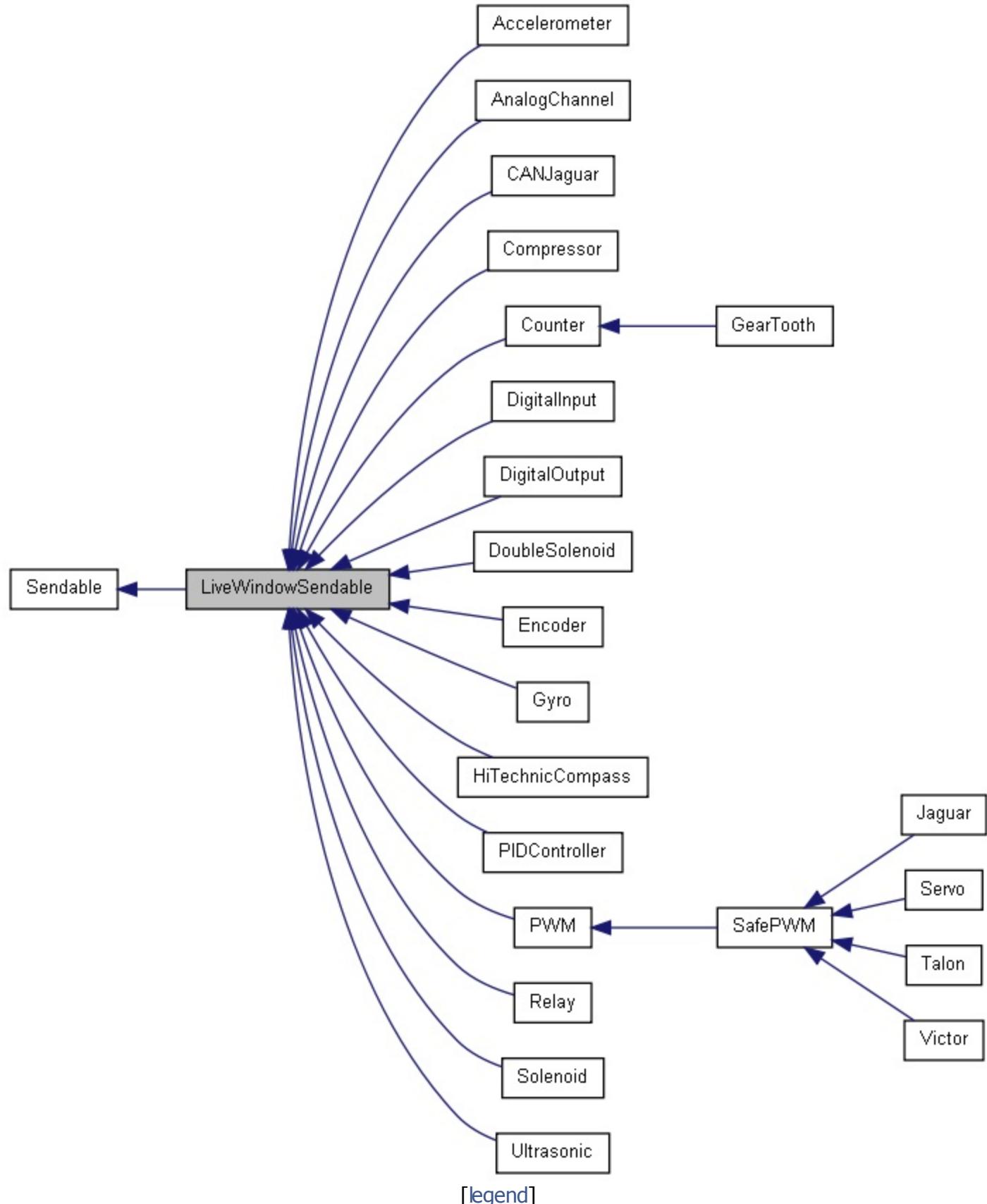


# LiveWindowSendable Class Reference

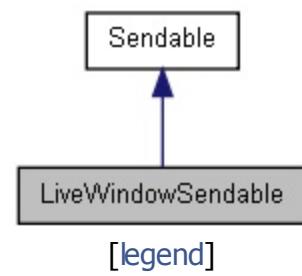
Live Window **Sendable** is a special type of object sendable to the live window. More...

```
#include <LiveWindowSendable.h>
```

Inheritance diagram for LiveWindowSendable:



## Collaboration diagram for LiveWindowSendable:



List of all members.

## Public Member Functions

virtual void **UpdateTable ()=0**

Update the table for this sendable object with the latest values.

virtual void **StartLiveWindowMode ()=0**

Start having this sendable object automatically respond to value changes reflect the value on the table.

virtual void **StopLiveWindowMode ()=0**

Stop having this sendable object automatically respond to value changes.

---

# Detailed Description

Live Window **Sendable** is a special type of object sendable to the live window.

## Author:

Patrick Plenefisch

---

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

- C:/WindRiver/workspace/WPILib/LiveWindow/**LiveWindowSendable.h**

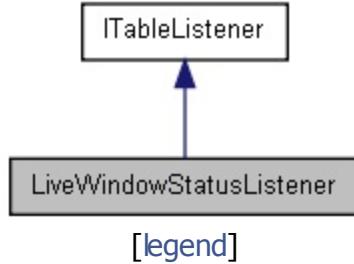
---

Generated by  1.7.2

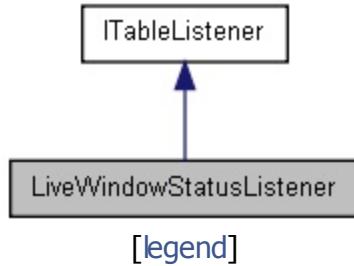


# LiveWindowStatusListener Class Reference

Inheritance diagram for LiveWindowStatusListener:



Collaboration diagram for LiveWindowStatusListener:



List of all members.

## Public Member Functions

virtual void **ValueChanged (ITable \*source, const std::string &key, EntryValue value, bool isNew)**

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

---

# Member Function Documentation

```
virtual void LiveWindowStatusListener::ValueChanged ( ITable *  
                                                 const std::string & key  
                                                 EntryValue value  
                                                 bool isNew  
                                                 ) [v]
```

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

## Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

---

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

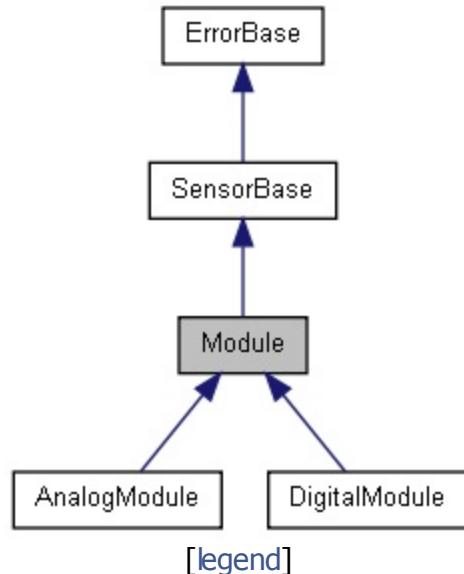
- C:/WindRiver/workspace/WPILib/LiveWindow/**LiveWindowStatusListener.h**

[Class List](#)[Class Hierarchy](#)[Class Members](#)

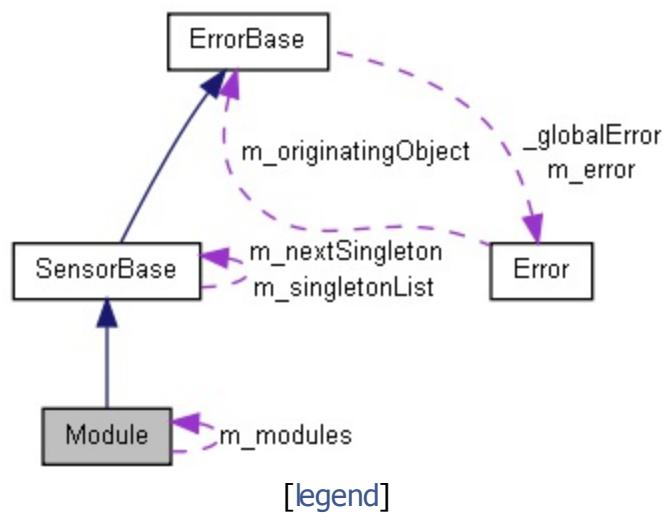
[Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#)

# Module Class Reference

Inheritance diagram for Module:



Collaboration diagram for Module:



List of all members.

# Public Member Functions

---

nLoadOut::tModuleType **GetType ()**  
    UINT8 **GetNumber ()**

static **Module** \* **GetModule** (nLoadOut::tModuleType type, UINT8 number)  
    Static module singleton factory.

**Module** (nLoadOut::tModuleType type, UINT8 number)

Constructor.

virtual **~Module ()**

Destructor.

## Protected Attributes

nLoadOut::tModuleType **m\_moduleType**

The type of module represented.

UINT8 **m\_moduleNumber**

The module index within the module type.

# Constructor & Destructor Documentation

```
Module::Module( nLoadOut::tModuleType type,  
                UINT8                  number  
            )  
                           [explicit, protected]
```

Constructor.

## Parameters:

- type** The type of module represented.
- number** The module index within the module type.

# Member Function Documentation

```
Module * Module::GetModule( nLoadOut::tModuleType type,
                           UINT8                number
                           )
                           [static]
```

Static module singleton factory.

## Parameters:

- type** The type of module represented.
- number** The module index within the module type.

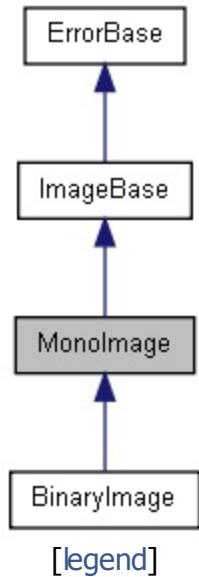
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/[Module.h](#)
- C:/WindRiver/workspace/WPILib/Module.cpp

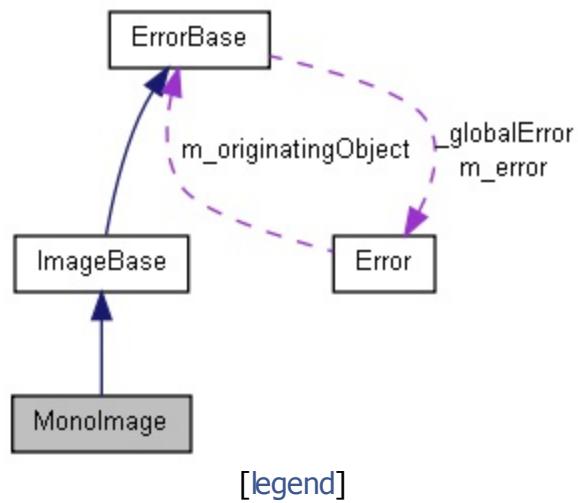


# MonoImage Class Reference

Inheritance diagram for MonoImage:



Collaboration diagram for MonoImage:



List of all members.

## Public Member Functions

vector< EllipseMatch > \* **DetectEllipses** (EllipseDescriptor \*ellipseDescriptor, CurveOptions \*curveOptions, ShapeDetectionOptions \*shapeDetectionOptions, ROI \*roi)  
Look for ellipses in an image.

vector< EllipseMatch > \* **DetectEllipses** (EllipseDescriptor \*ellipseDescriptor)

# Member Function Documentation

```
vector< EllipseMatch > * MonoImage::DetectEllipses ( EllipseDescriptor *  
                                                 CurveOptions *  
                                                 ShapeDetectionOptions  
                                                 ROI *  
                                                 )
```

Look for ellipses in an image.

Given some input parameters, look for any number of ellipses in an image.

## Parameters:

<b>ellipseDescriptor</b>	Ellipse descriptor
<b>curveOptions</b>	Curve options
<b>shapeDetectionOptions</b>	Shape detection options
<b>roi</b>	Region of Interest

## Returns:

a vector of EllipseMatch structures (0 length vector on no match)

---

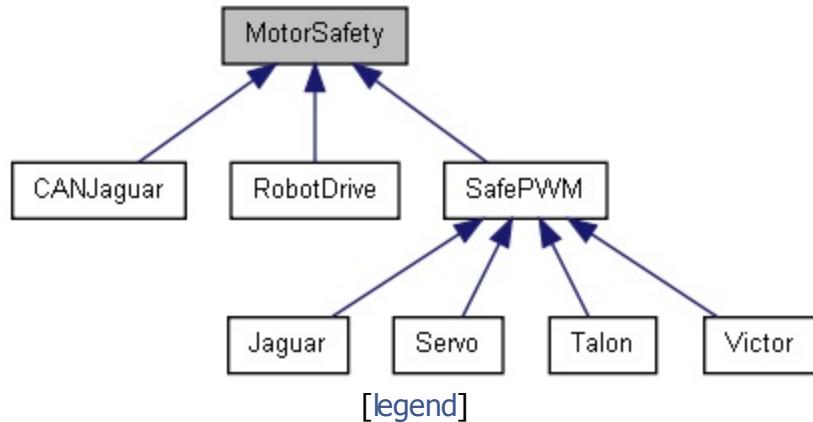
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/Vision/**MonoImage.h**
- C:/WindRiver/workspace/WPILib/Vision/MonoImage.cpp



# MotorSafety Class Reference

Inheritance diagram for MotorSafety:



List of all members.

## Public Member Functions

---

virtual void	<b>SetExpiration</b> (float timeout)=0
virtual float	<b>GetExpiration</b> ()=0
virtual bool	<b>IsAlive</b> ()=0
virtual void	<b>StopMotor</b> ()=0
virtual void	<b>SetSafetyEnabled</b> (bool enabled)=0
virtual bool	<b>IsSafetyEnabled</b> ()=0
virtual void	<b>GetDescription</b> (char *desc)=0

---

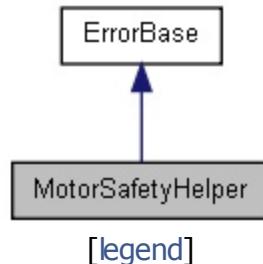
The documentation for this class was generated from the following file:

- C:/WindRiver/workspace/WPIlib/**MotorSafety.h**

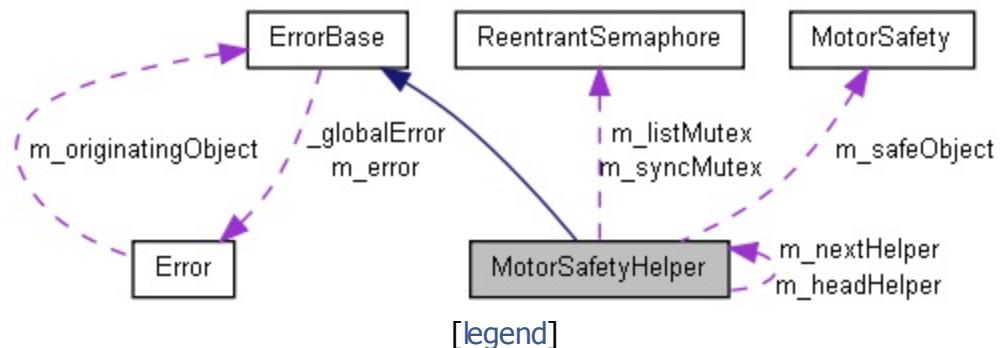
[Public Member Functions](#) |  
[Static Public Member Functions](#)

# MotorSafetyHelper Class Reference

Inheritance diagram for MotorSafetyHelper:



Collaboration diagram for MotorSafetyHelper:



List of all members.

# Public Member Functions

**MotorSafetyHelper (MotorSafety \*safeObject)**

The constructor for a **MotorSafetyHelper** object.

void **Feed ()**

void **SetExpiration (float expirationTime)**

float **GetExpiration ()**

Retrieve the timeout value for the corresponding motor safety object.

bool **IsAlive ()**

Determine if the motor is still operating or has timed out.

void **Check ()**

Check if this motor has exceeded its timeout.

void **SetSafetyEnabled (bool enabled)**

Enable/disable motor safety for this device Turn on and off the motor safety option for this **PWM** object.

bool **IsSafetyEnabled ()**

Return the state of the motor safety enabled flag Return if the motor safety is currently enabled for this device.

static void **CheckMotors ()**

Check the motors to see if any have timed out.

# Constructor & Destructor Documentation

## **MotorSafetyHelper::MotorSafetyHelper ( MotorSafety \* safeObject )**

The constructor for a **MotorSafetyHelper** object.

The helper object is constructed for every object that wants to implement the Motor Safety protocol. The helper object has the code to actually do the timing and call the motors Stop() method when the timeout expires. The motor object is expected to call the Feed() method whenever the motors value is updated.

### **Parameters:**

**safeObject** a pointer to the motor object implementing **MotorSafety**. This is used to call the Stop() method on the motor.

# Member Function Documentation

## **void MotorSafetyHelper::Check( )**

Check if this motor has exceeded its timeout.

This method is called periodically to determine if this motor has exceeded its timeout value. If it has, the stop method is called, and the motor is shut down until its value is updated again.

## **void MotorSafetyHelper::CheckMotors( ) [static]**

Check the motors to see if any have timed out.

This static method is called periodically to poll all the motors and stop any that have timed out.

## **float MotorSafetyHelper::GetExpiration( )**

Retrieve the timeout value for the corresponding motor safety object.

### **Returns:**

the timeout value in seconds.

## **bool MotorSafetyHelper::IsAlive( )**

Determine if the motor is still operating or has timed out.

### **Returns:**

a true value if the motor is still operating normally and hasn't timed out.

## **bool MotorSafetyHelper::IsSafetyEnabled( )**

Return the state of the motor safety enabled flag Return if the motor safety is currently enabled for this device.

### **Returns:**

True if motor safety is enforced for this device

## **void MotorSafetyHelper::SetSafetyEnabled ( bool enabled )**

Enable/disable motor safety for this device Turn on and off the motor safety option for this **PWM** object.

### **Parameters:**

**enabled** True if motor safety is enforced for this object

---

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

- C:/WindRiver/workspace/WPILib/**MotorSafetyHelper.h**
- C:/WindRiver/workspace/WPILib/MotorSafetyHelper.cpp

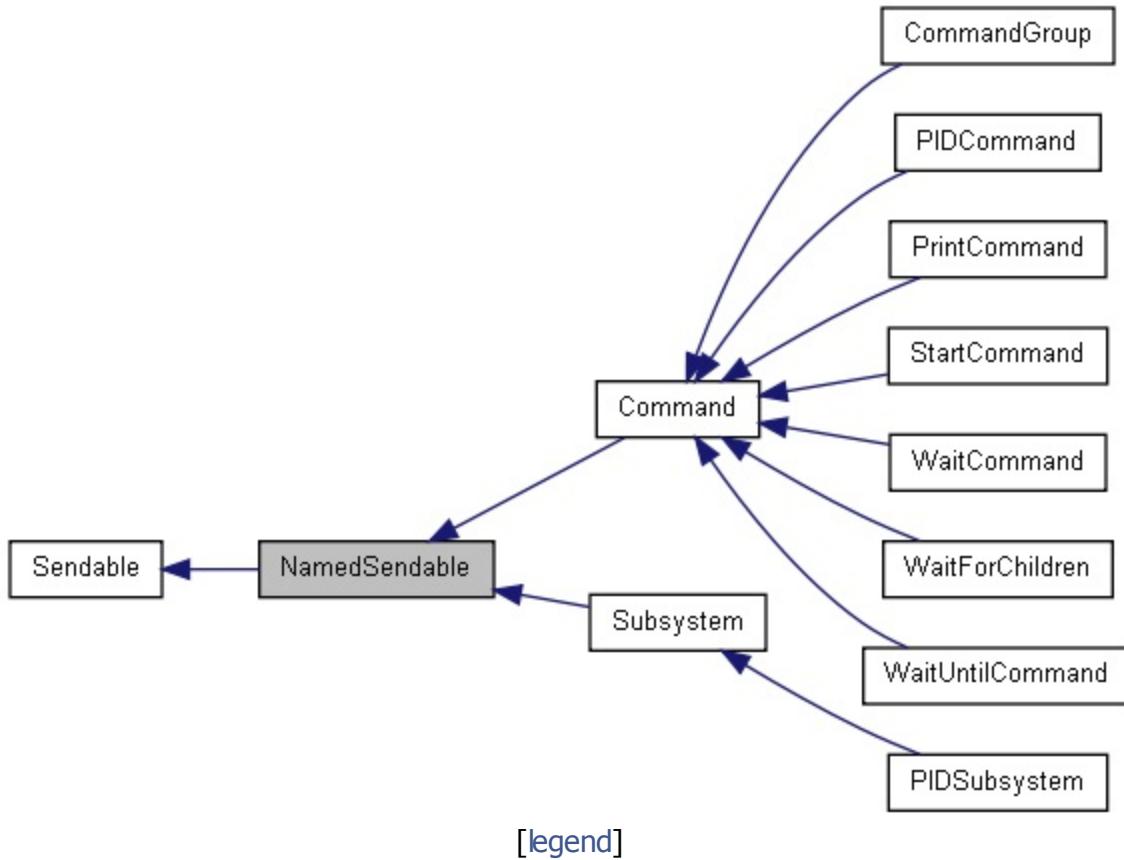


# NamedSendable Class Reference

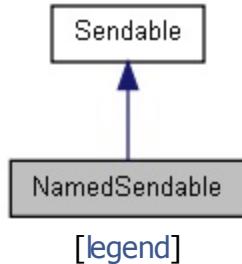
The interface for sendable objects that gives the sendable a default name in the Smart Dashboard. More...

```
#include <NamedSendable.h>
```

Inheritance diagram for NamedSendable:



Collaboration diagram for NamedSendable:



List of all members.

## Public Member Functions

```
virtual std::string GetName ()=0
```

---

## Detailed Description

The interface for sendable objects that gives the sendable a default name in the [Smart Dashboard](#).

---

# Member Function Documentation

**virtual std::string NamedSendable::GetName( ) [pure virtual]**

## Returns:

the name of the subtable of **SmartDashboard** that the **Sendable** object will use

Implemented in **Command**, and **Subsystem**.

---

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

- C:/WindRiver/workspace/WPILib/SmartDashboard/**NamedSendable.h**

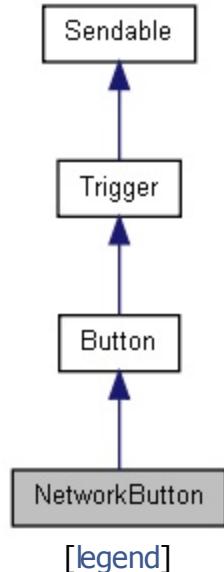
---

Generated by  1.7.2

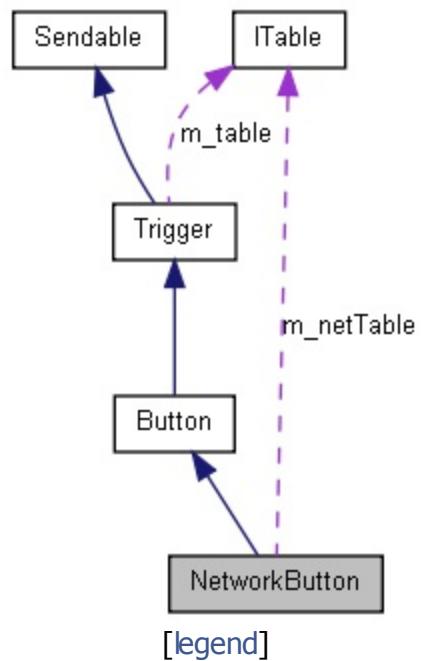


# NetworkButton Class Reference

Inheritance diagram for NetworkButton:



Collaboration diagram for NetworkButton:



List of all members.

# Public Member Functions

**NetworkButton** (const char \*tableName, const char \*field)

**NetworkButton** (**ITable** \*table, const char \*field)

virtual bool **Get** ()

---

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

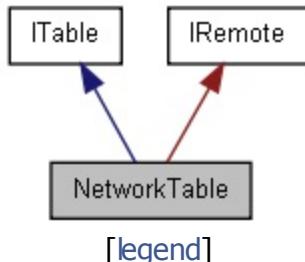
- C:/WindRiver/workspace/WPILib/Buttons/**NetworkButton.h**
- C:/WindRiver/workspace/WPILib/Buttons/NetworkButton.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

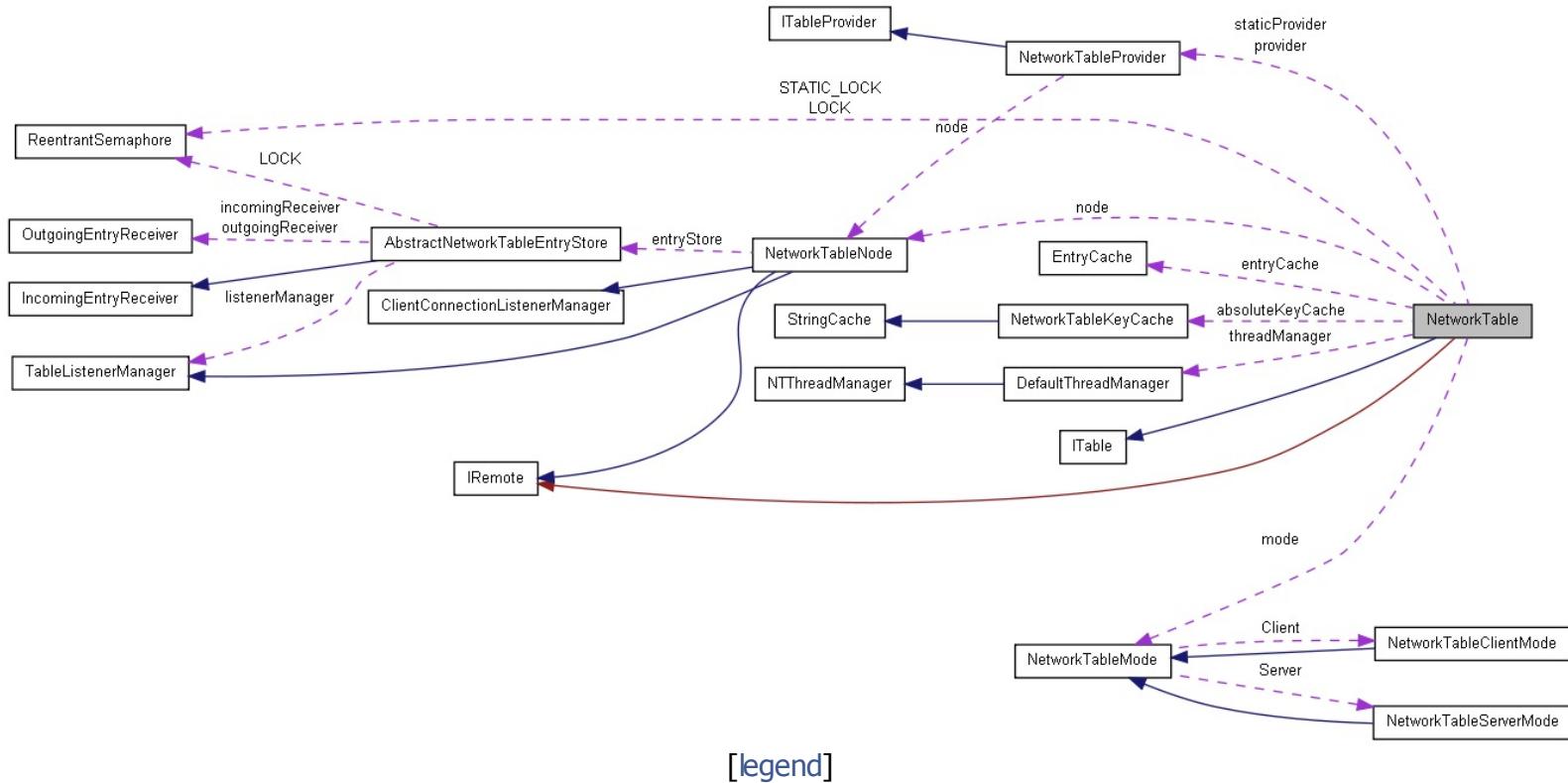
[Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Static Public Attributes](#)

# NetworkTable Class Reference

## Inheritance diagram for NetworkTable:



## Collaboration diagram for NetworkTable:



## List of all members.

**NetworkTable** (std::string path, **NetworkTableProvider**&provider)

bool **IsConnected** ()

Get the current state of the objects connection.

bool **IsServer** ()

If the object is acting as a server.

void

## AddConnectionListener (IRemoteConnectionListener)

\*Listener hook immediate Notify  
Register an object to listen to connection and disconnection events.

void	<b>RemoveConnectionListener</b> ( <b>IRemoteConnectionListener</b> *listener) Unregister a listener from connection events.
void	<b>AddTableListener</b> ( <b>ITableListener</b> *listener) Add a listener for changes to the table.
void	<b>AddTableListener</b> ( <b>ITableListener</b> *listener, bool immediateNotify) Add a listener for changes to the table.
void	<b>AddTableListener</b> (std::string key, <b>ITableListener</b> *listener, bool immediateNotify) Add a listener for changes to a specific key the table.
void	<b>AddSubTableListener</b> ( <b>ITableListener</b> *listener) This will immediately notify the listener of all current sub tables.
void	<b>RemoveTableListener</b> ( <b>ITableListener</b> *listener) Remove a listener from receiving table events.
<b>NetworkTable</b> *	<b>GetSubTable</b> (std::string key) Returns the table at the specified key.
bool	<b>ContainsKey</b> (std::string key) Checks the table and tells if it contains the specified key.
bool	<b>ContainsSubTable</b> (std::string key)
void	<b>PutNumber</b> (std::string key, double value) Maps the specified key to the specified value in this table.
double	<b>GetNumber</b> (std::string key) Returns the key that the name maps to.
double	<b>GetNumber</b> (std::string key, double defaultValue) Returns the key that the name maps to.
void	<b>PutString</b> (std::string key, std::string value) Maps the specified key to the specified value in this table.
std::string	<b>GetString</b> (std::string key)  Returns the key that the name maps to.
std::string	<b>GetString</b> (std::string key, std::string defaultValue) Returns the key that the name maps to.
void	<b>PutBoolean</b> (std::string key, bool value) Maps the specified key to the specified value in this table.
bool	<b>GetBoolean</b> (std::string key)

bool	Returns the key that the name maps to. <b>GetBoolean</b> (std::string key, bool defaultValue)
void	Returns the key that the name maps to. <b>PutValue</b> (std::string key, NetworkTableEntryType *type, EntryValue value)
void	<b>RetrieveValue</b> (std::string key, ComplexData &externalValue)
void	<b>PutValue</b> (std::string key, ComplexData &value) Maps the specified key to the specified value in this table.
<b>EntryValue</b>	<b>GetValue</b> (std::string key) Returns the key that the name maps to.
<b>EntryValue</b>	<b>GetValue</b> (std::string key, EntryValue defaultValue) Returns the key that the name maps to.

static void	<b>Initialize</b> ()
static void	<b>SetServerMode</b> () set that network tables should be a server This must be called before initialize or GetTable
static void	<b>SetTeam</b> (int team)

set the team the robot is configured for (this will set the ip address that network tables will connect to in client mode)  
This must be called before initialize or GetTable

static void **SetIPAddress** (const char \*address)

static **NetworkTable** \* **GetTable** (std::string key)

Gets the table with the specified key.

static const char **PATH\_SEPARATOR\_CHAR** = '/'

static const std::string **PATH\_SEPARATOR**

The path separator for sub-tables and keys.

static const int **DEFAULT\_PORT** = 1735

The default port that network tables operates on.

# Member Function Documentation

```
void NetworkTable::AddConnectionListener ( IRemoteConnectionListener * lis  
                                         bool  
                                         )
```

Register an object to listen for connection and disconnection events.

## Parameters:

**listener** the listener to be register

**immediateNotify** if the listener object should be notified of the current connection state

Implements **IRemote**.

```
void NetworkTable::AddSubTableListener ( ITableListener * listener ) [virtual]
```

This will immediately notify the listener of all current sub tables.

## Parameters:

**listener**

Implements **ITable**.

```
void NetworkTable::AddTableListener ( ITableListener * listener ) [virtual]
```

Add a listener for changes to the table.

## Parameters:

**listener** the listener to add

Implements **ITable**.

```
void NetworkTable::AddTableListener ( ITableListener * listener,  
                                         bool  
                                         immediateNotify  
                                         )
```

[virtual]

Add a listener for changes to the table.

**Parameters:**

**listener** the listener to add  
**immediateNotify** if true then this listener will be notified of all current entries (marked as new)

Implements **ITable**.

```
void NetworkTable::AddTableListener( std::string      key,  
                                    ITableListener * listener,  
                                    bool            immediateNotify  
                                )  
                                [virtual]
```

Add a listener for changes to a specific key the table.

**Parameters:**

**key** the key to listen for  
**listener** the listener to add  
**immediateNotify** if true then this listener will be notified of all current entries (marked as new)

Implements **ITable**.

```
bool NetworkTable::ContainsKey( std::string key ) [virtual]
```

Checks the table and tells if it contains the specified key.

**Parameters:**

**key** the key to be checked

Implements **ITable**.

```
bool NetworkTable::ContainsSubTable( std::string key ) [virtual]
```

**Parameters:**

**key** the key to search for

**Returns:**

true if there is a subtable with the key which contains at least one key/subtable of its own

Implements **ITable**.

## **bool NetworkTable::GetBoolean ( std::string key ) [virtual]**

Returns the key that the name maps to.

### **Parameters:**

**key** the key name

### **Returns:**

the key

### **Exceptions:**

**TableKeyNotDefinedException** if the specified key is null

Implements **ITable**.

## **bool NetworkTable::GetBoolean ( std::string key,                                   bool       defaultValue                                   )                          [virtual]**

Returns the key that the name maps to.

If the key is null, it will return the default value

### **Parameters:**

**key** the key name

**defaultValue** the default value if the key is null

### **Returns:**

the key

Implements **ITable**.

## **double NetworkTable::GetNumber ( std::string key ) [virtual]**

Returns the key that the name maps to.

### **Parameters:**

**key** the key name

**Returns:**

the key

**Exceptions:**

**TableKeyNotDefinedException** if the specified key is null

Implements **ITable**.

```
double NetworkTable::GetNumber ( std::string key,  
                                double      defaultValue  
                            )  
                           [virtual]
```

Returns the key that the name maps to.

If the key is null, it will return the default value

**Parameters:**

**key** the key name

**defaultValue** the default value if the key is null

**Returns:**

the key

Implements **ITable**.

```
std::string NetworkTable::GetString ( std::string key,  
                                      std::string defaultValue  
                                  )  
                                     [virtual]
```

Returns the key that the name maps to.

If the key is null, it will return the default value

**Parameters:**

**key** the key name

**defaultValue** the default value if the key is null

**Returns:**

the key

Implements **ITable**.

## **std::string NetworkTable::GetString ( std::string key ) [virtual]**

Returns the key that the name maps to.

### **Parameters:**

**key** the key name

### **Returns:**

the key

### **Exceptions:**

**TableKeyNotDefinedException** if the specified key is null

Implements **ITable**.

## **NetworkTable \* NetworkTable::GetSubTable ( std::string key ) [virtual]**

Returns the table at the specified key.

If there is no table at the specified key, it will create a new table

### **Parameters:**

**key** the key name

### **Returns:**

the networktable to be returned

Implements **ITable**.

## **NetworkTable \* NetworkTable::GetTable ( std::string key ) [static]**

Gets the table with the specified key.

If the table does not exist, a new table will be created.

This will automatically initialize network tables if it has not been already

### **Parameters:**

**key** the key name

### **Returns:**

the network table requested

```
EntryValue NetworkTable::GetValue ( std::string key,
                                EntryValue defaultValue
)
```

Returns the key that the name maps to.

If the key is null, it will return the default value NOTE: If the value is a double, it will return a Double object, not a primitive. To get the primitive, use GetDouble

**Parameters:**

**key** the key name

**defaultValue** the default value if the key is null

**Returns:**

the key

```
EntryValue NetworkTable::GetValue ( std::string key ) [virtual]
```

Returns the key that the name maps to.

NOTE: If the value is a double, it will return a Double object, not a primitive. To get the primitive, use GetDouble

**Parameters:**

**key** the key name

**Returns:**

the key

**Exceptions:**

**TableKeyNotDefinedException** if the specified key is null

Implements **ITable**.

```
void NetworkTable::Initialize ( ) [static]
```

**Exceptions:**

**IOException**

## **bool NetworkTable::IsConnected ( )** [virtual]

Get the current state of the objects connection.

### **Returns:**

the current connection state

Implements **IRemote**.

## **bool NetworkTable::IsServer ( )** [virtual]

If the object is acting as a server.

### **Returns:**

if the object is a server

Implements **IRemote**.

## **void NetworkTable::PutBoolean ( std::string key,                                 bool         value                                 )** [virtual]

Maps the specified key to the specified value in this table.

The key can not be null. The value can be retrieved by calling the get method with a key that is equal to the original key.

### **Parameters:**

**key** the key

**value** the value

Implements **ITable**.

## **void NetworkTable::PutNumber ( std::string key,                                 double      value                                 )** [virtual]

Maps the specified key to the specified value in this table.

The key can not be null. The value can be retrieved by calling the get method with a key that is equal to the original key.

#### Parameters:

**key** the key  
**value** the value

Implements **ITable**.

```
void NetworkTable::PutString ( std::string key,  
                               std::string value  
                           ) [virtual]
```

Maps the specified key to the specified value in this table.

The key can not be null. The value can be retrieved by calling the get method with a key that is equal to the original key.

#### Parameters:

**key** the key  
**value** the value

Implements **ITable**.

```
void NetworkTable::PutValue ( std::string key,  
                             ComplexData & value  
                           ) [virtual]
```

Maps the specified key to the specified value in this table.

The key can not be null. The value can be retrieved by calling the get method with a key that is equal to the original key.

#### Parameters:

**key** the key name  
**value** the value to be put

Implements **ITable**.

## **void NetworkTable::RemoveConnectionListener ( IRemoteConnectionListener**

Unregister a listener from connection events.

### **Parameters:**

**listener** the listener to be unregistered

Implements **IRemote**.

## **void NetworkTable::RemoveTableListener ( ITableListener \* listener ) [virtual]**

Remove a listener from receiving table events.

### **Parameters:**

**listener** the listener to be removed

Implements **ITable**.

## **void NetworkTable::SetIPAddress ( const char \* address ) [static]**

### **Parameters:**

**address** the address that network tables will connect to in client mode

## **void NetworkTable::SetTeam ( int team ) [static]**

set the team the robot is configured for (this will set the ip address that network tables will connect to in client mode) This must be called before initialize or GetTable

### **Parameters:**

**team** the team number

# Member Data Documentation

**const std::string NetworkTable::PATH\_SEPARATOR [static]**

The path separator for sub-tables and keys.

---

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

- C:/WindRiver/workspace/WPIlib/networktables/**NetworkTable.h**
- C:/WindRiver/workspace/WPIlib/networktables/NetworkTable.cpp

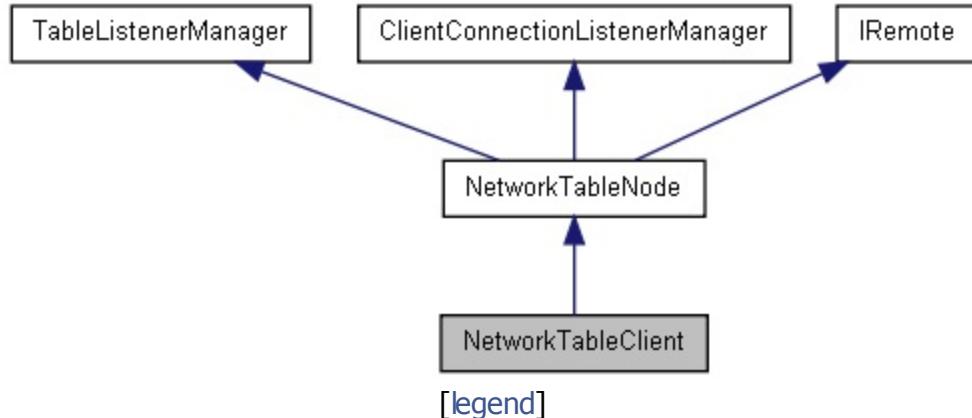


# NetworkTableClient Class Reference

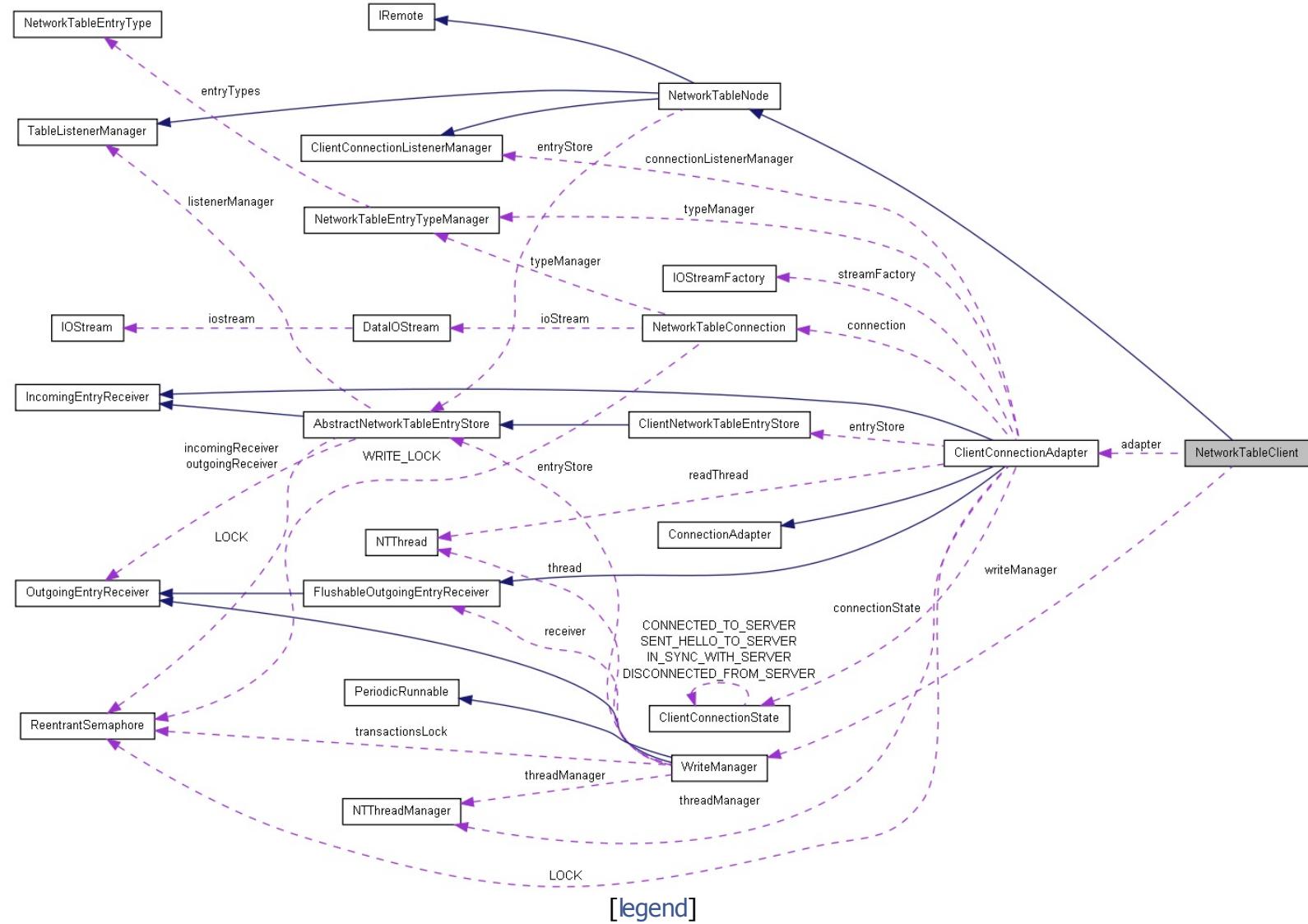
A client node in NetworkTables 2.0. More...

```
#include <NetworkTableClient.h>
```

Inheritance diagram for NetworkTableClient:



Collaboration diagram for NetworkTableClient:



List of all members.

---

**NetworkTableClient** (**IOStreamFactory** &streamFactory,  
**NetworkTableEntryTypeManager** &typeManager, **NTThreadManager**  
&threadManager)

Create a new **NetworkTable** Client.

---

**void reconnect ()**

force the client to disconnect and reconnect to the server again.

---

**void Close ()**

close all networking activity related to this node

---

**void stop ()**

**bool IsConnected ()**

Get the current state of the objects connection.

---

**bool IsServer ()**

If the object is acting as a server.

---

# Detailed Description

A client node in NetworkTables 2.0.

## Author:

Mitchell

---

# Constructor & Destructor Documentation

**NetworkTableClient::NetworkTableClient( IOStreamFactory &  
NetworkTableEntryTypeManager &  
NTThreadManager &  
)**

Create a new **NetworkTable** Client.

## Parameters:

**streamFactory**

**threadManager**

**transactionPool**

# Member Function Documentation

## **bool NetworkTableClient::IsConnected( ) [virtual]**

Get the current state of the objects connection.

### **Returns:**

the current connection state

Implements **IRemote**.

## **bool NetworkTableClient::IsServer( ) [virtual]**

If the object is acting as a server.

### **Returns:**

if the object is a server

Implements **IRemote**.

## **void NetworkTableClient::reconnect( )**

force the client to disconnect and reconnect to the server again.

Will connect if the client is currently disconnected

---

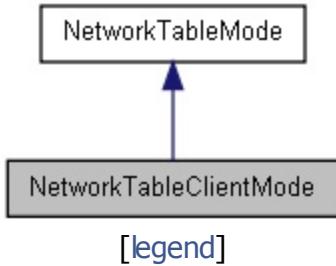
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/client/**NetworkTableClient.h**
- C:/WindRiver/workspace/WPILib/networktables2/client/NetworkTableClient.cpp

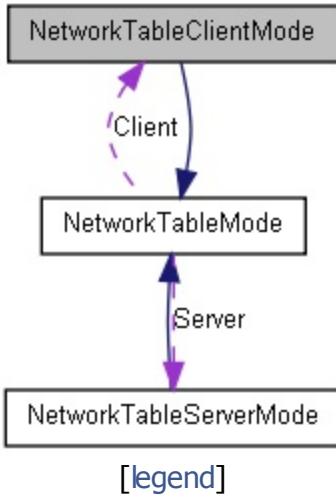


# NetworkTableClientMode Class Reference

Inheritance diagram for NetworkTableClientMode:



Collaboration diagram for NetworkTableClientMode:



List of all members.

## Public Member Functions

virtual **NetworkTableNode** \* **CreateNode** (const char \*ipAddress, int port,  
**NTThreadManager** &threadManager)

# Member Function Documentation

```
NetworkTableNode * NetworkTableClientMode::CreateNode ( const char *
                                                       int
                                                       NTTthreadManager
)

```

## Parameters:

**ipAddress** the IP address configured by the user  
**port** the port configured by the user  
**threadManager** the thread manager that should be used for threads in the node

## Returns:

a new node that can back a network table

## Exceptions:

**IOException**

Implements **NetworkTableMode**.

---

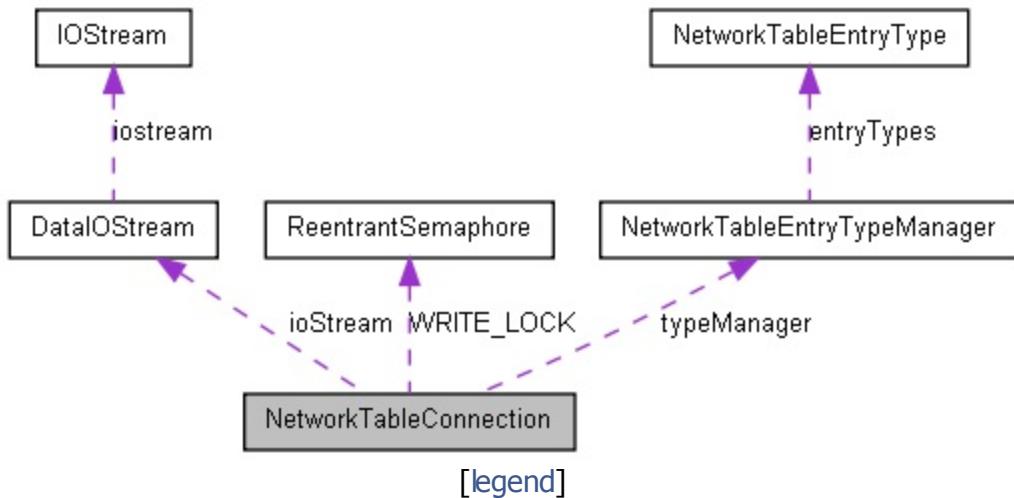
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTableMode.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTableMode.cpp



# NetworkTableConnection Class Reference

Collaboration diagram for NetworkTableConnection:



List of all members.

# Public Member Functions

**NetworkTableConnection (IOStream \*stream,  
NetworkTableEntryTypeManager &typeManager)**  
An abstraction for the **NetworkTable** protocol.

```
void close ()
void flush ()
void sendKeepAlive ()
void sendClientHello ()
void sendServerHelloComplete ()
void sendProtocolVersionUnsupported ()
void sendEntryAssignment (NetworkTableEntry &entry)
void sendEntryUpdate (NetworkTableEntry &entry)
void read (ConnectionAdapter &adapter)
```

---

static const ProtocolVersion **PROTOCOL\_REVISION** = 0x0200

# Constructor & Destructor Documentation

**NetworkTableConnection::NetworkTableConnection ( [IOStream](#) \*  
   [NetworkTableEntryType](#)  
   )**

An abstraction for the [NetworkTable](#) protocol.

## Author:

mwills

---

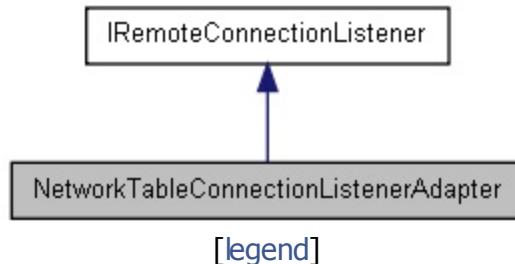
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/connection/[NetworkTableConnection.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/connection/NetworkTableConnection.cpp

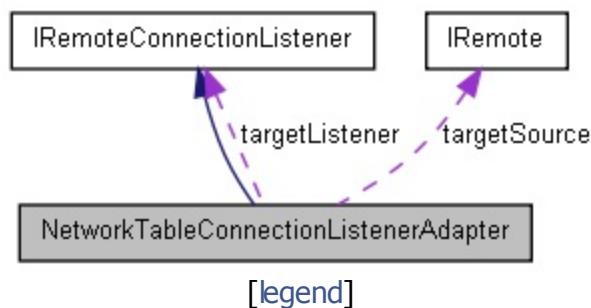


# NetworkTableConnectionListenerAdapter Class Reference

Inheritance diagram for NetworkTableConnectionListenerAdapter:



Collaboration diagram for NetworkTableConnectionListenerAdapter:



List of all members.

## Public Member Functions

**NetworkTableConnectionListenerAdapter (IRemote \*targetSource, IRemoteConnectionListener \*targetListener)**

void **Connected (IRemote \*remote)**

Called when an **IRemote** is connected.

void **Disconnected (IRemote \*remote)**

Called when an **IRemote** is disconnected.

# Member Function Documentation

## **void NetworkTableConnectionListenerAdapter::Connected ( IRemote \* remote )**

Called when an **IRemote** is connected.

### Parameters:

**remote** the object that connected

Implements **IRemoteConnectionListener**.

## **void NetworkTableConnectionListenerAdapter::Disconnected ( IRemote \* remote )**

Called when an **IRemote** is disconnected.

### Parameters:

**remote** the object that disconnected

Implements **IRemoteConnectionListener**.

---

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

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTableConnectionListenerAdapter.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTableConnectionListenerAdap

[Class List](#)[Class Hierarchy](#)[Class Members](#)

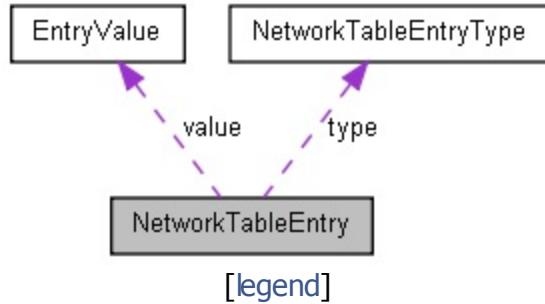
[Public Member Functions](#) | [Public Attributes](#) |  
[Static Public Attributes](#)

# NetworkTableEntry Class Reference

An entry in a network table. [More...](#)

```
#include <NetworkTableEntry.h>
```

Collaboration diagram for NetworkTableEntry:



[List of all members.](#)

# Public Member Functions

**NetworkTableEntry** (std::string &name,  
**NetworkTableEntryType** \*type, **EntryValue** value)

**NetworkTableEntry** (EntryId id, std::string &name,  
SequenceNumber sequenceNumber,  
**NetworkTableEntryType** \*type, **EntryValue** value)

EntryId **GetId** ()

**EntryValue** **GetValue** ()

**NetworkTableEntryType** \* **GetType** ()

bool **PutValue** (SequenceNumber newSequenceNumber,  
**EntryValue** newValue)

void **ForcePut** (SequenceNumber newSequenceNumber,  
**EntryValue** newValue)

force a value and new sequence number upon an entry

void **ForcePut** (SequenceNumber newSequenceNumber,  
**NetworkTableEntryType** \*type, **EntryValue**

newValue)

force a value and new sequence number upon an  
entry, Will also set the type of the entry

void **MakeDirty** ()

void **MakeClean** ()

bool **IsDirty** ()

void **SendValue** (**DataIOStream** &iostream)

Send the value of the entry over the output stream.

SequenceNumber **GetSequenceNumber** ()

void **SetId** (EntryId id)

Sets the id of the entry.

void **ClearId** ()

clear the id of the entry to unknown

void **Send** (**NetworkTableConnection** &connection)

void **FireListener** (**TableListenerManager**  
&listenerManager)

## Public Attributes

---

std::string **name**  
the name of the entry

## Static Public Attributes

---

static const EntryId **UNKNOWN\_ID** = 0xFFFF  
the id that represents that an id is unknown for an entry

## Detailed Description

An entry in a network table.

### **Author:**

mwills

---

# Member Function Documentation

```
void NetworkTableEntry::ForcePut( SequenceNumber           newSequenceNumber
                                  NetworkTableEntryType * newType,
                                  EntryValue*             newValue
                                )
```

force a value and new sequence number upon an entry, Will also set the type of the entry

## Parameters:

**newSequenceNumber**  
**type**  
**newValue**

```
void NetworkTableEntry::ForcePut( SequenceNumber newSequenceNumber,
                                  EntryValue*       newValue
                                )
```

force a value and new sequence number upon an entry

## Parameters:

**newSequenceNumber**  
**newValue**

**SequenceNumber NetworkTableEntry::GetSequenceNumber( )**

## Returns:

the current sequence number of the entry

**void NetworkTableEntry::SendValue( DataIOSStream & iostream )**

Send the value of the entry over the output stream.

## Parameters:

**os**

## Exceptions:

## **void NetworkTableEntry::SetId ( EntryId \_id )**

Sets the id of the entry.

### **Parameters:**

**id** the id of the entry

### **Exceptions:**

**IllegalStateException** if the entry already has a known id

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

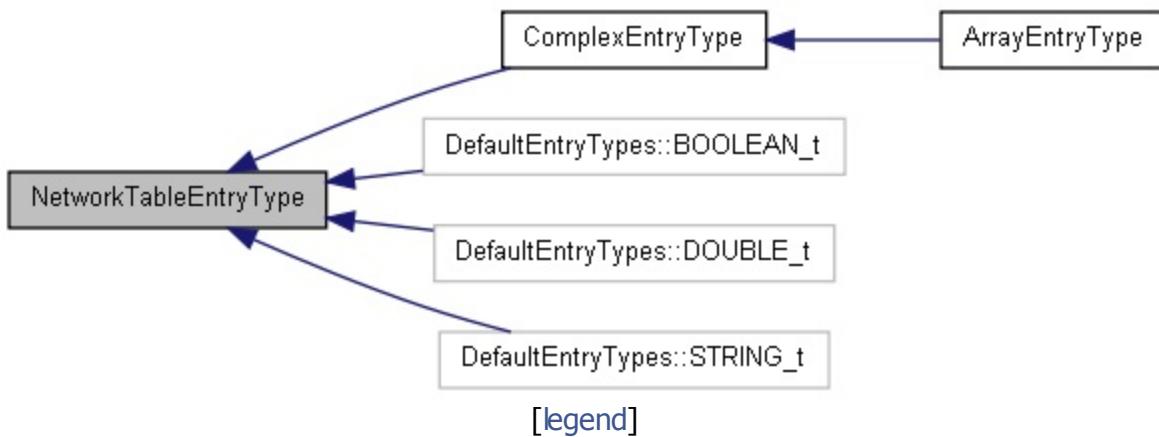
- C:/WindRiver/workspace/WPILib/networktables2/**NetworkTableEntry.h**
- C:/WindRiver/workspace/WPILib/networktables2/NetworkTableEntry.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

[Public Member Functions](#) | [Public Attributes](#) |  
[Protected Member Functions](#)

# NetworkTableEntryType Class Reference

Inheritance diagram for NetworkTableEntryType:



List of all members.

## Public Member Functions

---

virtual bool **isComplex** ()

virtual void **sendValue** (**EntryValue** value, **DataIOSream** &os)=0

virtual **EntryValue** **readValue** (**DataIOSream** &is)=0

virtual **EntryValue** **copyValue** (**EntryValue** value)

virtual void **deleteValue** (**EntryValue** value)

---

const TypeId **id**

const char \* **name**

# Protected Member Functions

---

## **NetworkTableEntryType** (TypeId id, const char \*name)

---

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

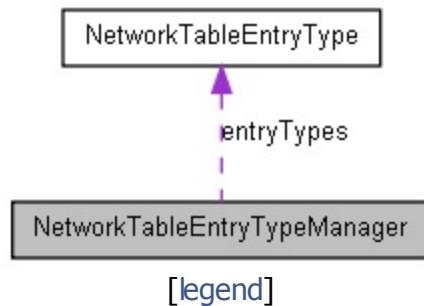
- C:/WindRiver/workspace/WPILib/networktables2/type/**NetworkTableEntryType.h**
- C:/WindRiver/workspace/WPILib/networktables2/type/NetworkTableEntryType.cpp

Generated by  1.7.2



# NetworkTableEntryTypeManager Class Reference

Collaboration diagram for NetworkTableEntryTypeManager:



[List of all members.](#)

# Public Member Functions

---

**NetworkTableEntryType** \* **GetType** (TypeId type)

void **RegisterType** (**NetworkTableEntryType** &type)

---

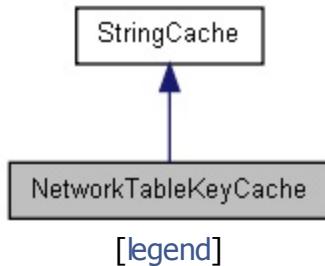
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/type/**NetworkTableEntryTypeM...**
- C:/WindRiver/workspace/WPILib/networktables2/type/NetworkTableEntryTypeManag...

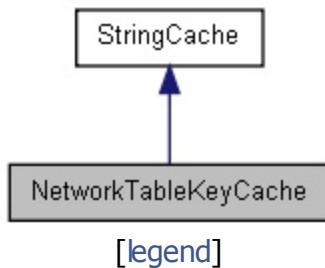


# NetworkTableKeyCache Class Reference

Inheritance diagram for NetworkTableKeyCache:



Collaboration diagram for NetworkTableKeyCache:



List of all members.

# Public Member Functions

**NetworkTableKeyCache** (std::string path)

std::string **Calc** (const std::string &key)

Will only be called if a value has not already been calculated.

---

# Member Function Documentation

**std::string NetworkTableKeyCache::Calc ( const std::string & **input** ) [virtual]**

Will only be called if a value has not already been calculated.

## Parameters:

**input**

## Returns:

the calculated value for a given input

Implements **StringCache**.

---

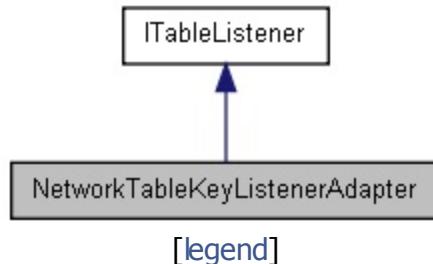
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPIlib/networktables/**NetworkTable.h**
- C:/WindRiver/workspace/WPIlib/networktables/NetworkTable.cpp

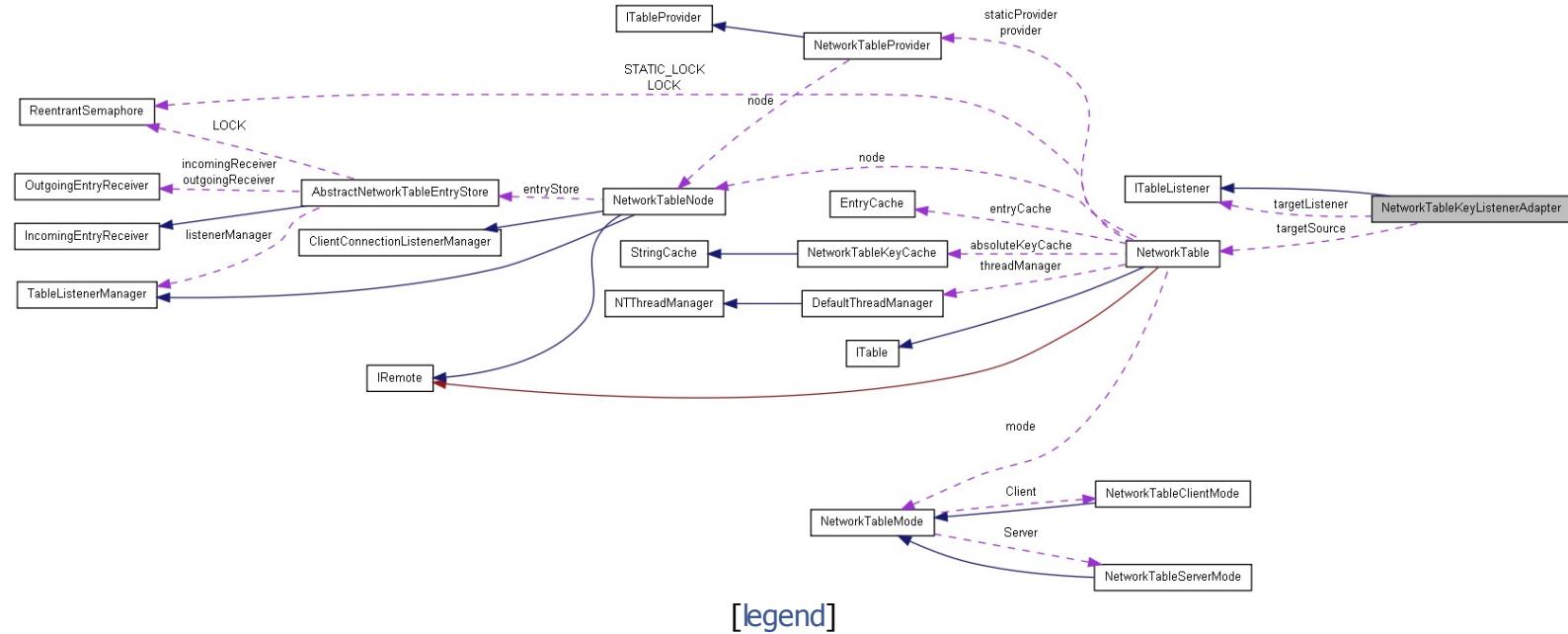


# NetworkTableKeyListenerAdapter Class Reference

Inheritance diagram for NetworkTableKeyListenerAdapter:



Collaboration diagram for NetworkTableKeyListenerAdapter:



List of all members.

**NetworkTableKeyListenerAdapter** (std::string relativeKey, std::string fullKey, **NetworkTable** \*targetSource, **ITableListener** \*targetListener)

void **ValueChanged** (**ITable** \*source, const std::string &key, **EntryValue** value, bool isNew)

Called when a key-value pair is changed in a **ITable**. WARNING: If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

# Member Function Documentation

```
void NetworkTableKeyListenerAdapter::ValueChanged ( ITable *  
                                                 const std::string & ke
```

```
EntryValue[
```

```
bool
```

```
)
```

```
so  
ke  
va  
is  
[v
```

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

## Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

---

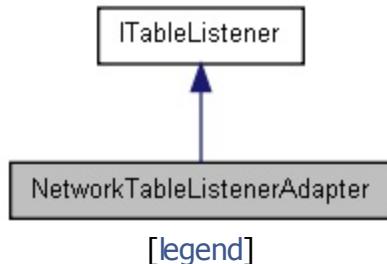
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTableKeyListenerAdapter.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTableKeyListenerAdapter.cpp

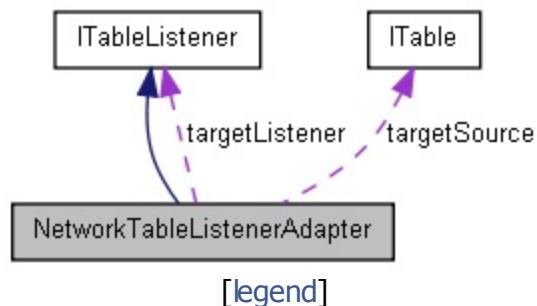


# NetworkTableListenerAdapter Class Reference

Inheritance diagram for NetworkTableListenerAdapter:



Collaboration diagram for NetworkTableListenerAdapter:



List of all members.

# Public Member Functions

**NetworkTableListenerAdapter** (std::string prefix, **ITable** \*targetSource,  
**ITableListener** \*targetListener)

void **ValueChanged** (**ITable** \*source, const std::string &key, **EntryValue** value,  
bool isNew)

Called when a key-value pair is changed in a **ITable** **WARNING:** If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

# Member Function Documentation

```
void NetworkTableListenerAdapter::ValueChanged( ITable *  
                                              const std::string &  
                                              EntryValue  
                                              bool  
                                              )  
                                              [virtu
```

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

## Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

---

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

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTableListenerAdapter.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTableListenerAdapter.cpp

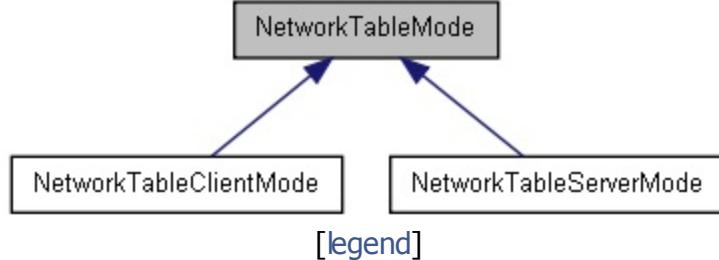


# NetworkTableMode Class Reference

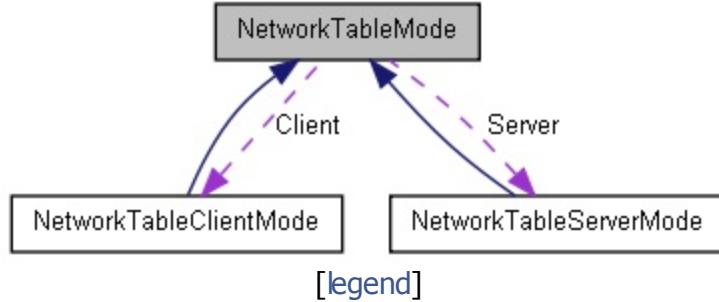
Represents a different modes that network tables can be configured in. [More...](#)

```
#include <NetworkTableMode.h>
```

Inheritance diagram for NetworkTableMode:



Collaboration diagram for NetworkTableMode:



[List of all members.](#)

## Public Member Functions

virtual **NetworkTableNode** \* **CreateNode** (const char \*ipAddress, int port,  
**NTThreadManager** &threadManager)=0

static **NetworkTableServerMode** **Server**

static **NetworkTableClientMode** **Client**

## Detailed Description

Represents a different modes that network tables can be configured in.

### Author:

Mitchell

---

# Member Function Documentation

```
virtual NetworkTableNode* NetworkTableMode::CreateNode ( const char *
int
NTThreadManager
)
```

## Parameters:

**ipAddress** the IP address configured by the user  
**port** the port configured by the user  
**threadManager** the thread manager that should be used for threads in the node

## Returns:

a new node that can back a network table

## Exceptions:

**IOException**

Implemented in **NetworkTableServerMode**, and **NetworkTableClientMode**.

---

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

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTableMode.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTableMode.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

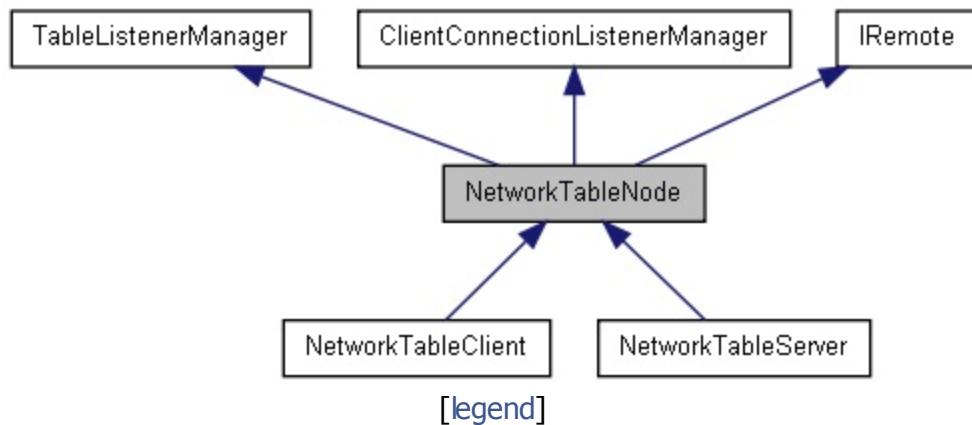
[Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#)

# NetworkTableNode Class Reference

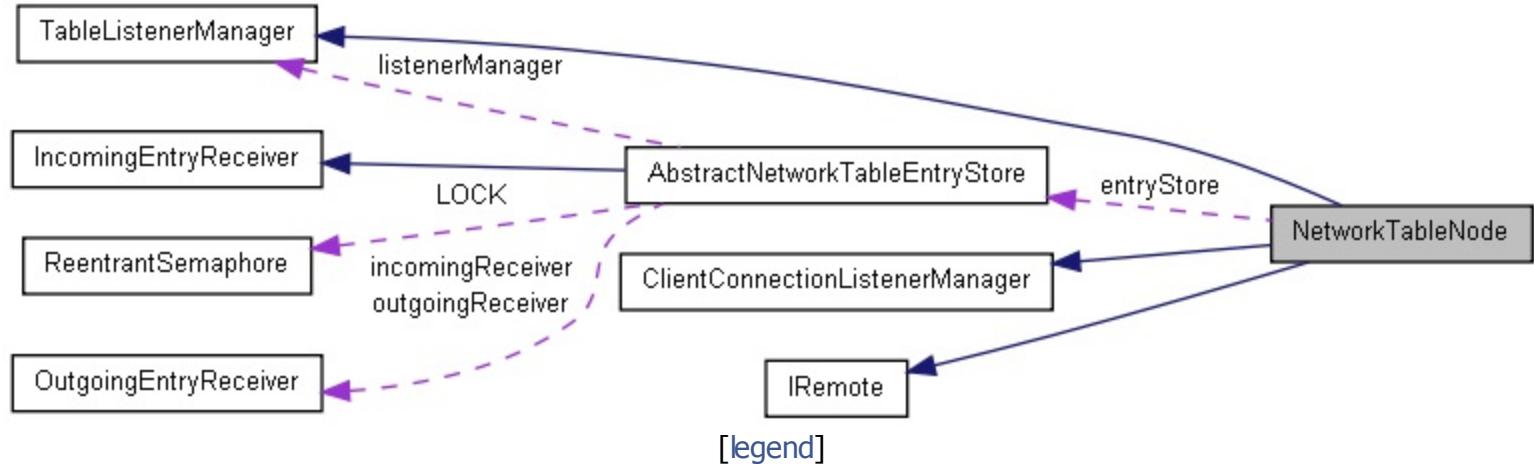
represents a node (either a client or a server) in a network tables 2.0  
implementers of the class must ensure that they call  
**init(NetworkTableTransactionPool, AbstractNetworkTableEntryStore)** before  
calling any other methods on this class [More...](#)

```
#include <NetworkTableNode.h>
```

Inheritance diagram for NetworkTableNode:



Collaboration diagram for NetworkTableNode:



List of all members.

# Public Member Functions

<b>AbstractNetworkTableEntryStore</b> &	<b>GetEntryStore ()</b>
void	<b>PutBoolean</b> (std::string &name, bool value)
bool	<b>GetBoolean</b> (std::string &name)
void	<b>PutDouble</b> (std::string &name, double value)
double	<b>GetDouble</b> (std::string &name)
void	<b>PutString</b> (std::string &name, std::string &value)
std::string &	<b>GetString</b> (std::string &name)
void	<b>PutComplex</b> (std::string &name, <b>ComplexData</b> &value)
void	<b>retrieveValue</b> (std::string &name, <b>ComplexData</b> &externalData)
void	<b>PutValue</b> (std::string &name, <b>NetworkTableEntryType</b> *type, <b>EntryValue</b> value)
	Put a value with a specific network table type.
void	<b>PutValue</b> ( <b>NetworkTableEntry</b> *entry, <b>EntryValue</b> value)
<b>EntryValue</b>	<b>GetValue</b> (std::string &name)
bool	<b>ContainsKey</b> (std::string &key)
virtual void	<b>Close ()=0</b> close all networking activity related to this node
	<b>AddConnectionListener</b>
void	( <b>IRemoteConnectionListener</b> *listener, bool immediateNotify) Register an object to listen for connection and disconnection events.
	<b>RemoveConnectionListener</b>
void	( <b>IRemoteConnectionListener</b> *listener) Unregister a listener from connection events.
	<b>FireConnectedEvent ()</b> called when something is connected
void	<b>FireDisconnectedEvent ()</b> called when something is disconnected
	<b>AddTableListener</b> ( <b>ITableListener</b>

```
void *listener, bool immediateNotify)
void RemoveTableListener (ITableListener
*listener)
void FireTableListeners (std::string &key,
EntryValue value, bool isNew)
```

## **NetworkTableNode** (AbstractNetworkTableEntryStore &entryStore)

## Protected Attributes

**AbstractNetworkTableEntryStore & entryStore**

## Detailed Description

represents a node (either a client or a server) in a network tables 2.0

implementers of the class must ensure that they call

**init(NetworkTableTransactionPool, AbstractNetworkTableEntryStore)** before  
calling any other methods on this class

### Author:

Mitchell

---

# Member Function Documentation

```
void NetworkTableNode::AddConnectionListener( IRemoteConnectionListener< >
                                              bool
                                              )
```

Register an object to listen for connection and disconnection events.

## Parameters:

**listener** the listener to be register  
**immediateNotify** if the listener object should be notified of the current connection state

Implements **IRemote**.

```
bool NetworkTableNode::ContainsKey( std::string & key )
```

## Parameters:

**key** the key to check for existence

## Returns:

true if the table has the given key

```
AbstractNetworkTableEntryStore & NetworkTableNode::GetEntryStore( )
```

## Returns:

the entry store used by this node

```
void NetworkTableNode::PutValue( std::string & name,
                                 NetworkTableEntryType * type,
                                 EntryValue value
                                 )
```

Put a value with a specific network table type.

## Parameters:

**name** the name of the entry to associate with the given value  
**type** the type of the entry

**value** the actual value of the entry

## **void NetworkTableNode::RemoveConnectionListener ( IRemoteConnectionList**

Unregister a listener from connection events.

### **Parameters:**

**listener** the listener to be unregistered

Implements **IRemote**.

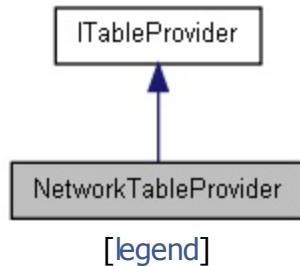
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/**NetworkTableNode.h**
- C:/WindRiver/workspace/WPILib/networktables2/NetworkTableNode.cpp



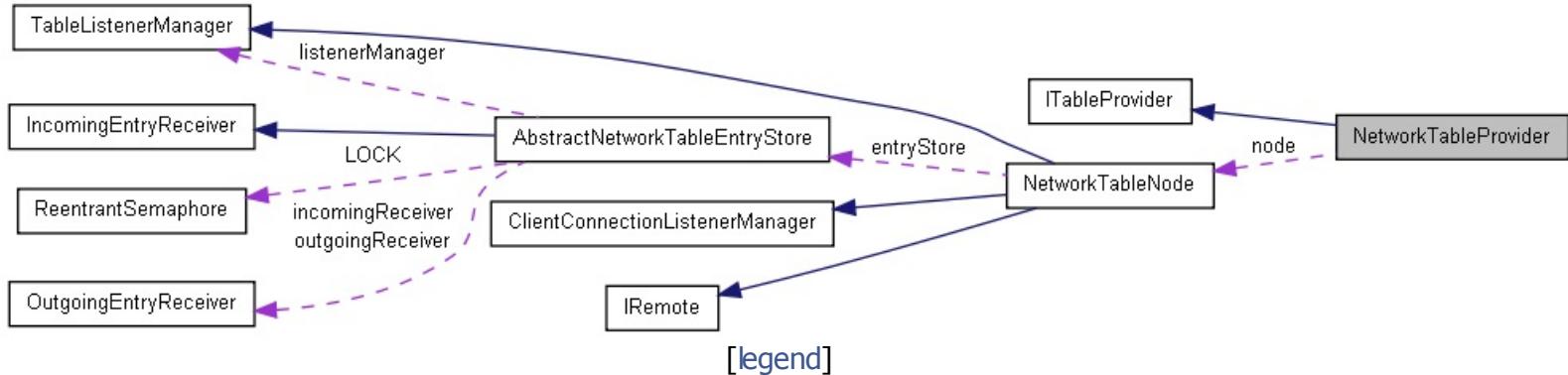
# NetworkTableProvider Class Reference

Inheritance diagram for NetworkTableProvider:



[legend]

Collaboration diagram for NetworkTableProvider:



[legend]

List of all members.

## Public Member Functions

**NetworkTableProvider** (**NetworkTreeNode** &node)

Create a new **NetworkTableProvider** for a given  
**NetworkTreeNode**.

**ITable** \* **GetRootTable** ()

**ITable** \* **GetTable** (std::string key)

Get a table by name.

**NetworkTreeNode** & **GetNode** ()

void **Close** ()

close the backing network table node

---

# Constructor & Destructor Documentation

## **NetworkTableProvider::NetworkTableProvider ( NetworkTreeNode & node )**

Create a new **NetworkTableProvider** for a given **NetworkTreeNode**.

### **Parameters:**

**node** the node that handles the actual network table

# Member Function Documentation

## **NetworkTableNode& NetworkTableProvider::GetNode( )** [inline]

### Returns:

the Network Table node that backs the Tables returned by this provider

## **ITable \* NetworkTableProvider::GetTable( std::string name )** [virtual]

Get a table by name.

### Parameters:

**name** the name of the table

### Returns:

a Table with the given name

Implements **ITableProvider**.

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

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTableProvider.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTableProvider.cpp

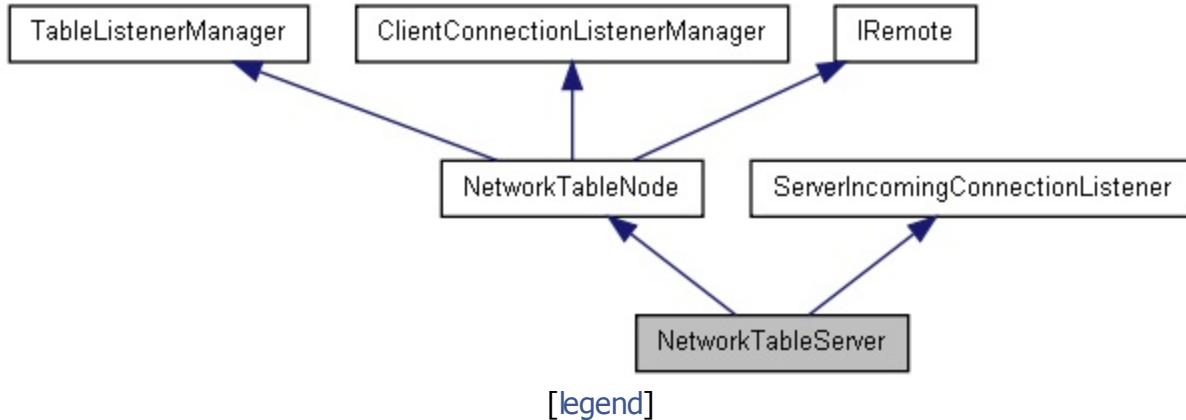


# NetworkTableServer Class Reference

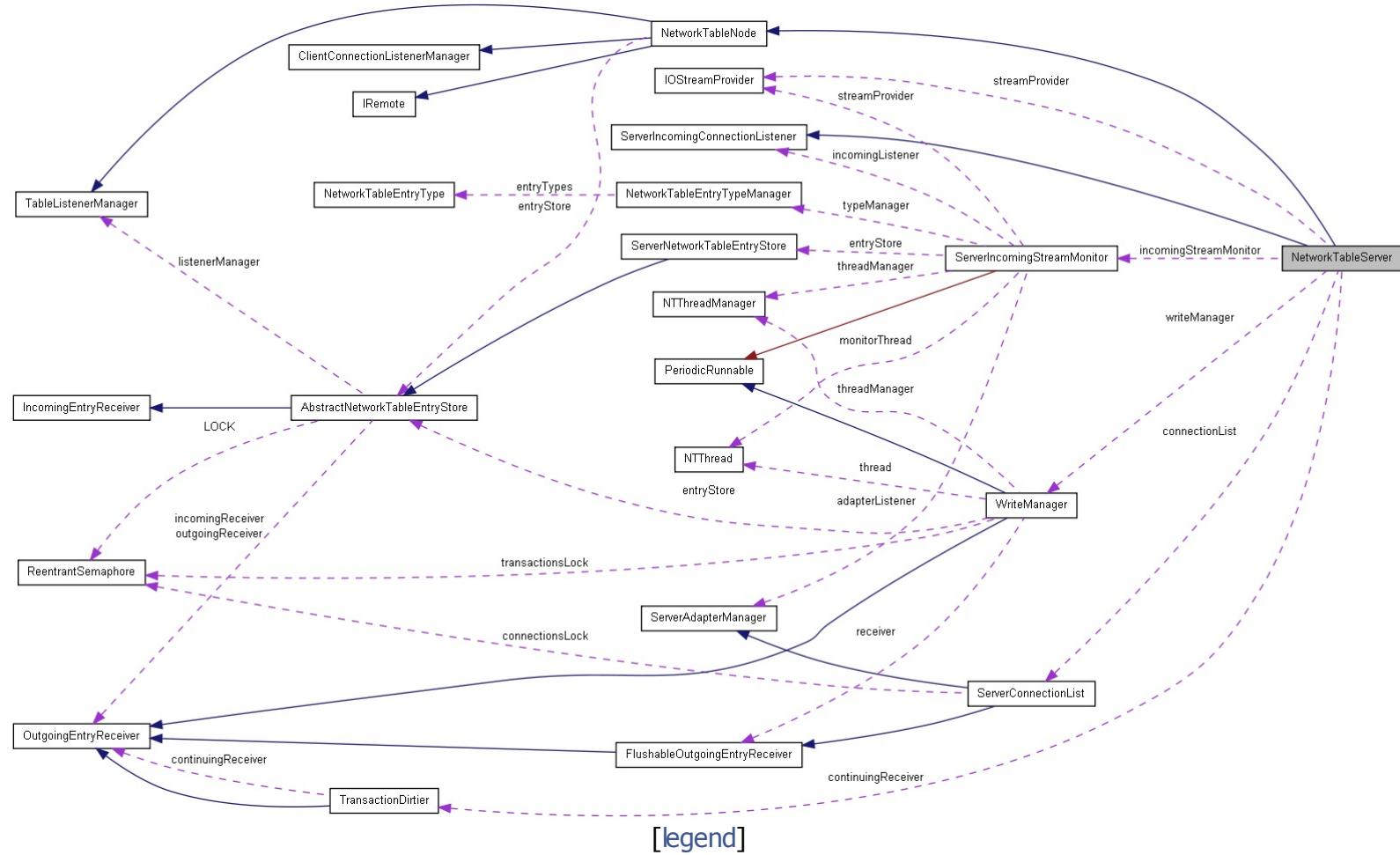
A server node in NetworkTables 2.0. More...

```
#include <NetworkTableServer.h>
```

Inheritance diagram for NetworkTableServer:



Collaboration diagram for NetworkTableServer:



List of all members.

## Public Member Functions

**NetworkTableServer** (**IOStreamProvider** &streamProvider,  
**NetworkTableEntryTypeManager** &typeManager, **NTThreadManager**  
&threadManager)

Create a **NetworkTable** Server.

**NetworkTableServer** (**IOStreamProvider** &streamProvider)

Create a **NetworkTable** Server.

void **Close** ()

close all networking activity related to this node

void **OnNewConnection** (**ServerConnectionAdapter** &connectionAdapter)

Called on create of a new connection.

bool **IsConnected** ()

Get the current state of the objects connection.

bool **IsServer** ()

If the object is acting as a server.

# Detailed Description

A server node in NetworkTables 2.0.

## Author:

Mitchell

---

# Constructor & Destructor Documentation

**NetworkTableServer::NetworkTableServer ( [IOStreamProvider](#) &  
                                 [NetworkTableEntryTypeManager](#)  
                                 [NTThreadManager](#) &  
                                 )**

Create a [NetworkTable](#) Server.

## Parameters:

**streamProvider  
threadManager  
transactionPool**

**NetworkTableServer::NetworkTableServer ( [IOStreamProvider](#) & [streamProvi](#)**

Create a [NetworkTable](#) Server.

## Parameters:

**streamProvider**

# Member Function Documentation

## **bool NetworkTableServer::IsConnected( ) [virtual]**

Get the current state of the objects connection.

### **Returns:**

the current connection state

Implements **IRemote**.

## **bool NetworkTableServer::IsServer( ) [virtual]**

If the object is acting as a server.

### **Returns:**

if the object is a server

Implements **IRemote**.

## **void NetworkTableServer::OnNewConnection( ServerConnectionAdapter & connectionAdapter ) [virtual]**

Called on create of a new connection.

### **Parameters:**

**connectionAdapter** the server connection adapter

Implements **ServerIncomingConnectionListener**.

---

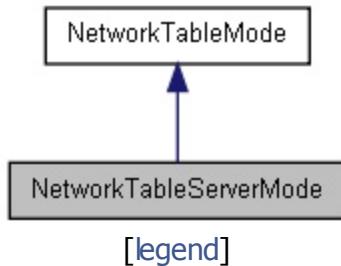
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables2/server/**NetworkTableServer.h**
- C:/WindRiver/workspace/WPILib/networktables2/server/NetworkTableServer.cpp

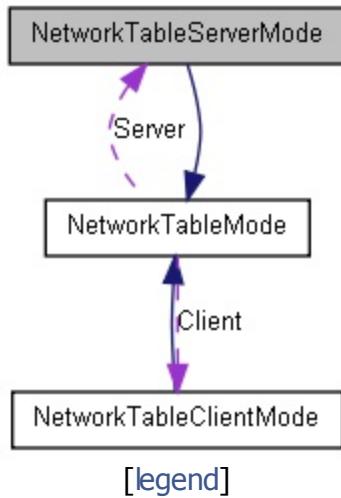


# NetworkTableServerMode Class Reference

Inheritance diagram for NetworkTableServerMode:



Collaboration diagram for NetworkTableServerMode:



List of all members.

## Public Member Functions

virtual **NetworkTableNode** \* **CreateNode** (const char \*ipAddress, int port,  
**NTThreadManager** &threadManager)

# Member Function Documentation

```
NetworkTableNode * NetworkTableServerMode::CreateNode ( const char *
                                                       int
                                                       NTThreadMana
)

```

## Parameters:

**ipAddress** the IP address configured by the user  
**port** the port configured by the user  
**threadManager** the thread manager that should be used for threads in the node

## Returns:

a new node that can back a network table

## Exceptions:

**IOException**

Implements **NetworkTableMode**.

---

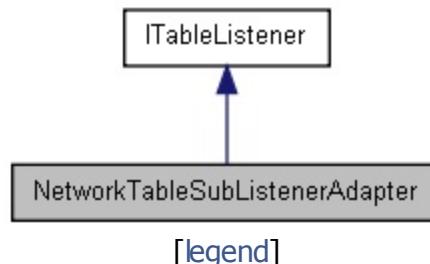
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTableMode.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTableMode.cpp



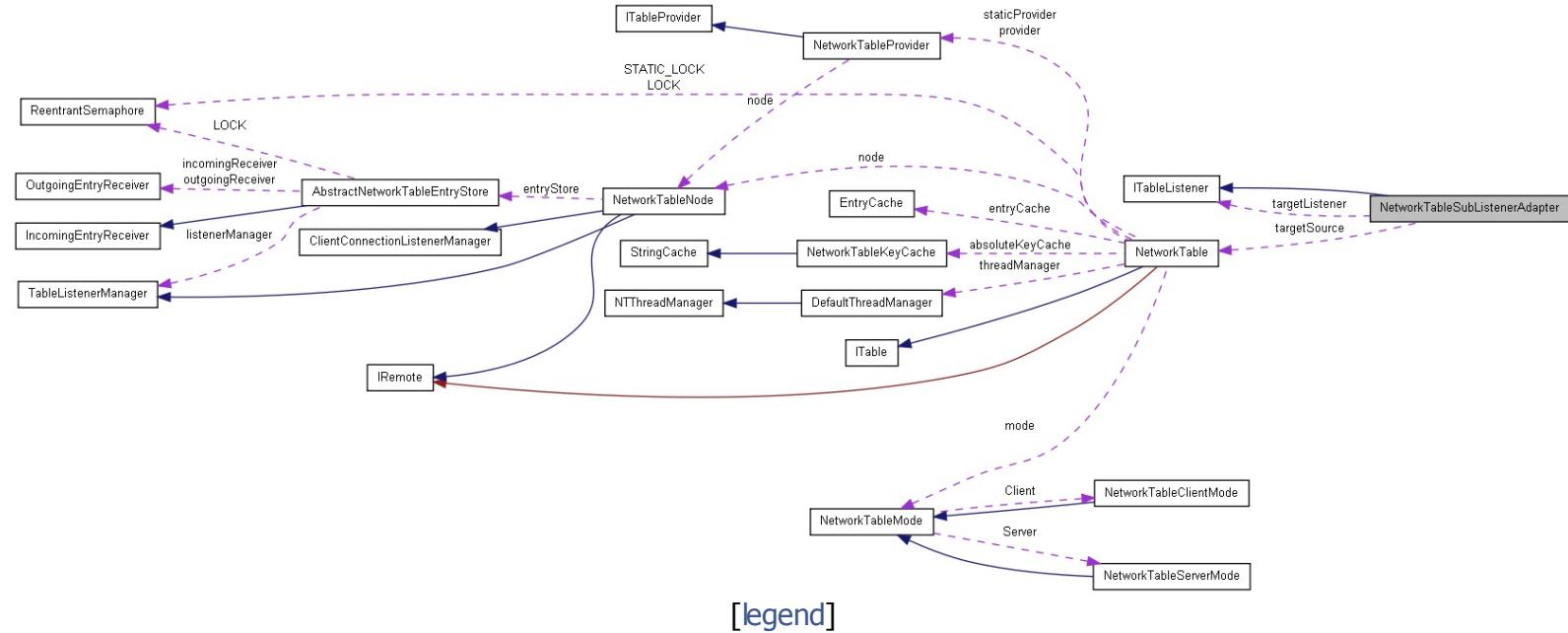
# NetworkTableSubListenerAdapter Class Reference

Inheritance diagram for NetworkTableSubListenerAdapter:



[legend]

Collaboration diagram for NetworkTableSubListenerAdapter:



[legend]

List of all members.

**NetworkTableSubListenerAdapter** (`std::string &prefix, NetworkTable *targetSource, ITableListener *targetListener)`)

Create a new adapter.

void

**ValueChanged** (`ITable *source, const std::string &key, EntryValue value, bool isNew)`)

Called when a key-value pair is changed in a **ITable** **WARNING:** If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

# Constructor & Destructor Documentation

**NetworkTableSubListenerAdapter::NetworkTableSubListenerAdapter ( std::string prefix, NetworkTable targetSource, ITable targetListener )**

Create a new adapter.

## Parameters:

**prefix** the prefix of the current table

**targetSource** the source that events passed to the target listener will appear to come from

**targetListener** the listener where events are forwarded to

# Member Function Documentation

```
void NetworkTableSubListenerAdapter::ValueChanged ( ITable *
```

```
          const std::string & key,
```

```
          EntryValue value,
```

```
          bool isNew)
```

```
) [v]
```

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

## Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

---

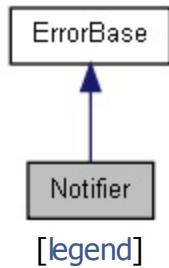
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/networktables/**NetworkTableSubListenerAdapter.h**
- C:/WindRiver/workspace/WPILib/networktables/NetworkTableSubListenerAdapter.cpp

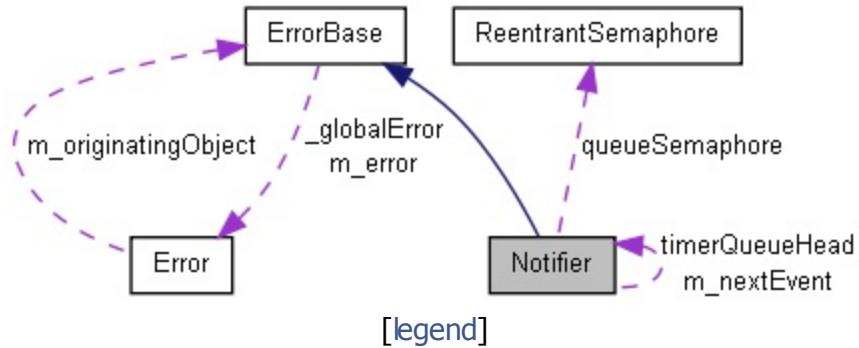


# Notifier Class Reference

Inheritance diagram for Notifier:



Collaboration diagram for Notifier:



List of all members.

# Public Member Functions

**Notifier** (TimerEventHandler handler, void \*param=NULL)

Create a **Notifier** for timer event notification.

virtual **~Notifier** ()

Free the resources for a timer event.

void **StartSingle** (double delay)

Register for single event notification.

void **StartPeriodic** (double period)

Register for periodic event notification.

void **Stop** ()

Stop timer events from occurring.

---

# Constructor & Destructor Documentation

```
Notifier::Notifier( TimerEventHandler handler,  
                    void * param = NULL  
)
```

Create a **Notifier** for timer event notification.

## Parameters:

**handler** The handler is called at the notification time which is set using StartSingle or StartPeriodic.

```
Notifier::~Notifier( ) [virtual]
```

Free the resources for a timer event.

All resources will be freed and the timer event will be removed from the queue if necessary.

# Member Function Documentation

## **void Notifier::StartPeriodic ( double period )**

Register for periodic event notification.

A timer event is queued for periodic event notification. Each time the interrupt occurs, the event will be immediately requeued for the same time interval.

### **Parameters:**

**period** Period in seconds to call the handler starting one period after the call to this method.

## **void Notifier::StartSingle ( double delay )**

Register for single event notification.

A timer event is queued for a single event after the specified delay.

### **Parameters:**

**delay** Seconds to wait before the handler is called.

## **void Notifier::Stop ( )**

Stop timer events from occurring.

Stop any repeating timer events from occurring. This will also remove any single notification events from the queue. If a timer-based call to the registered handler is in progress, this function will block until the handler call is complete.

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

- C:/WindRiver/workspace/WPILib/**Notifier.h**
- C:/WindRiver/workspace/WPILib/Notifier.cpp

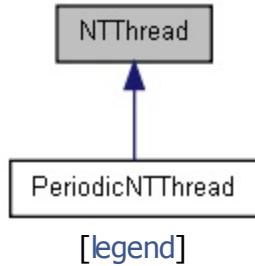


# NTThread Class Reference

Represents a thread in the network tables system. [More...](#)

```
#include <NTThread.h>
```

Inheritance diagram for NTThread:



[List of all members.](#)

## Public Member Functions

virtual void **stop** ()=0  
stop the thread

virtual bool **isRunning** ()=0

---

## Detailed Description

Represents a thread in the network tables system.

### Author:

mwills

---

# Member Function Documentation

**virtual bool NTThread::isRunning ( ) [pure virtual]**

## Returns:

true if the thread is running

Implemented in **PeriodicNTThread**.

---

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

- C:/WindRiver/workspace/WPILib/networktables2/thread/**NTThread.h**

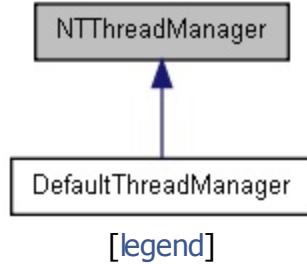


# NTThreadManager Class Reference

A thread manager that can be used to obtain new threads. [More...](#)

```
#include <NTThreadManager.h>
```

Inheritance diagram for NTThreadManager:



[List of all members.](#)

## Public Member Functions

```
virtual NTThread * newBlockingPeriodicThread (PeriodicRunnable *r, const  
char *name)=0
```

---

## Detailed Description

A thread manager that can be used to obtain new threads.

### **Author:**

Mitchell

---

# Member Function Documentation

```
virtual NTThread* NTThreadManager::newBlockingPeriodicThread ( PeriodicR
                                                               const cha
                                                               )
```

## Parameters:

r

**name** the name of the thread

## Returns:

a thread that will run the provided runnable repeatedly with the assumption that the runnable will block

---

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

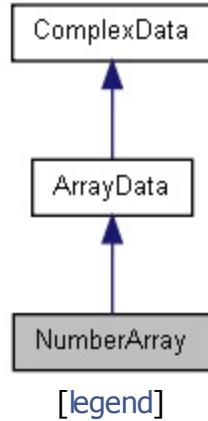
- C:/WindRiver/workspace/WPILib/networktables2/thread/**NTThreadManager.h**



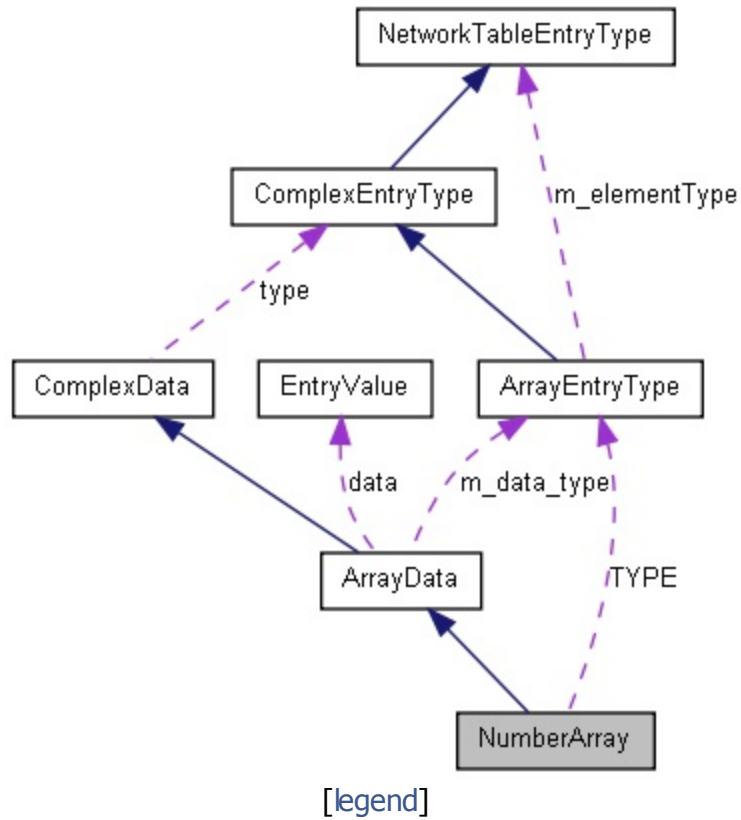
# NumberArray Class Reference

```
#include <NumberArray.h>
```

Inheritance diagram for NumberArray:



Collaboration diagram for NumberArray:



List of all members.

## Public Member Functions

```
double get (int index)
void set (int index, double value)
void add (double value)
```

```
static const TypeId NUMBER_ARRAY_RAW_ID = 0x11
```

```
static ArrayEntryType TYPE
```

---

# Detailed Description

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/networktables2/type/**NumberArray.h**
- C:/WindRiver/workspace/WPILib/networktables2/type/NumberArray.cpp

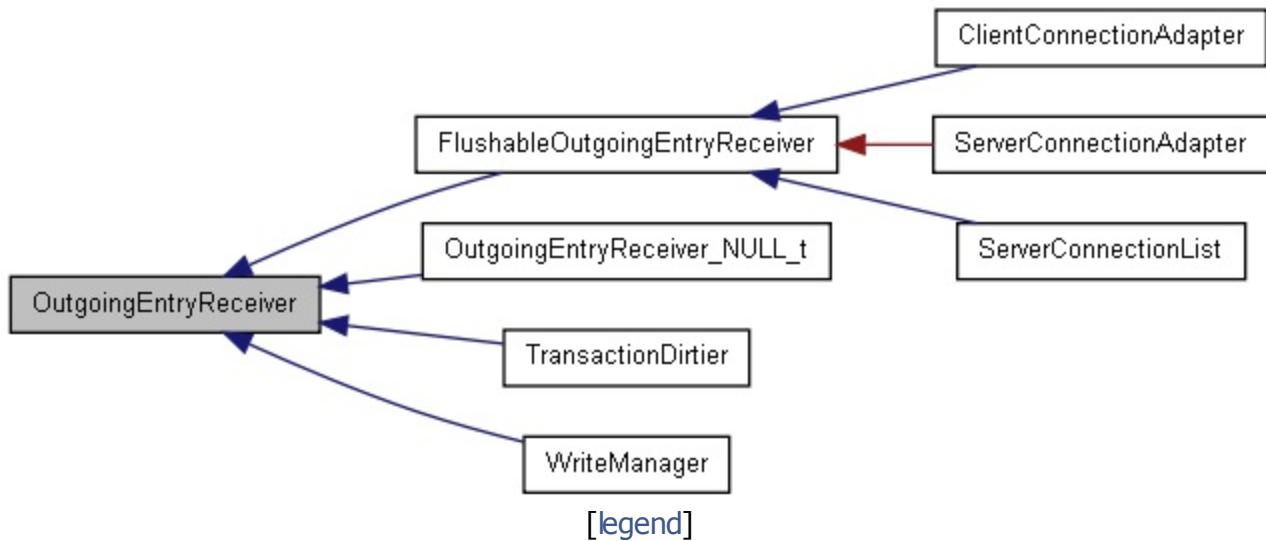
---

Generated by [doxygen](#) 1.7.2



# OutgoingEntryReceiver Class Reference

Inheritance diagram for OutgoingEntryReceiver:



List of all members.

## Public Member Functions

---

```
virtual void offerOutgoingAssignment (NetworkTableEntry *entry)=0
```

```
virtual void offerOutgoingUpdate (NetworkTableEntry *entry)=0
```

---

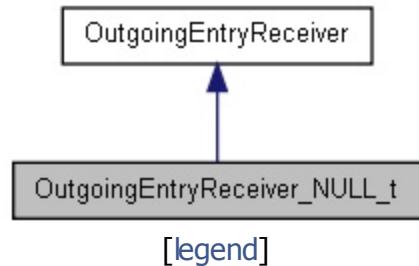
The documentation for this class was generated from the following file:

- C:/WindRiver/workspace/WPILib/networktables2/**OutgoingEntryReceiver.h**

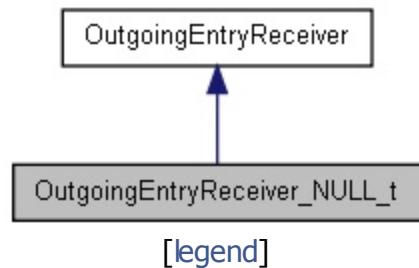


# OutgoingEntryReceiver\_NULL\_t Class Reference

Inheritance diagram for OutgoingEntryReceiver\_NULL\_t:



Collaboration diagram for OutgoingEntryReceiver\_NULL\_t:



List of all members.

## Public Member Functions

---

```
void offerOutgoingAssignment (NetworkTableEntry *entry)
void offerOutgoingUpdate (NetworkTableEntry *entry)
```

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

- C:/WindRiver/workspace/WPILib/networktables2/[OutgoingEntryReceiver.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/OutgoingEntryReciever.cpp



# ParticleAnalysisReport\_struct Struct Reference

---

frcParticleAnalysis returns this structure [More...](#)

```
#include <VisionAPI.h>
```

List of all members.

## Public Attributes

int	<b>imageHeight</b>
int	<b>imageWidth</b>
double	<b>imageTimestamp</b>
int	<b>particleIndex</b>
int	<b>center_mass_x</b>
int	<b>center_mass_y</b>
double	<b>center_mass_x_normalized</b>
double	<b>center_mass_y_normalized</b>
double	<b>particleArea</b>
Rect	<b>boundingRect</b>
double	<b>particleToImagePercent</b>
double	<b>particleQuality</b>

---

## Detailed Description

frcParticleAnalysis returns this structure

---

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

- C:/WindRiver/workspace/WPILib/Vision2009/**VisionAPI.h**



# **pcre\_callout\_block Struct Reference**

---

List of all members.

## Public Attributes

int	<b>version</b>
int	<b>callout_number</b>
int *	<b>offset_vector</b>
PCRE_SPTR	<b>subject</b>
int	<b>subject_length</b>
int	<b>start_match</b>
int	<b>current_position</b>
int	<b>capture_top</b>
int	<b>capture_last</b>
void *	<b>callout_data</b>
int	<b>pattern_position</b>
int	<b>next_item_length</b>

---

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

- C:/WindRiver/workspace/WPILib/[pcre.h](#)



# pcre\_extra Struct Reference

---

List of all members.

## Public Attributes

unsigned long int	<b>flags</b>
void *	<b>study_data</b>
unsigned long int	<b>match_limit</b>
void *	<b>callout_data</b>
const unsigned char *	<b>tables</b>
unsigned long int	<b>match_limit_recursion</b>

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

- C:/WindRiver/workspace/WPILib/[pcre.h](#)

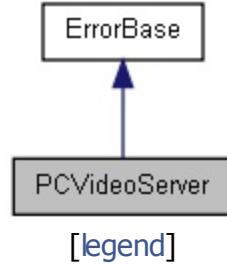


# PCVideoServer Class Reference

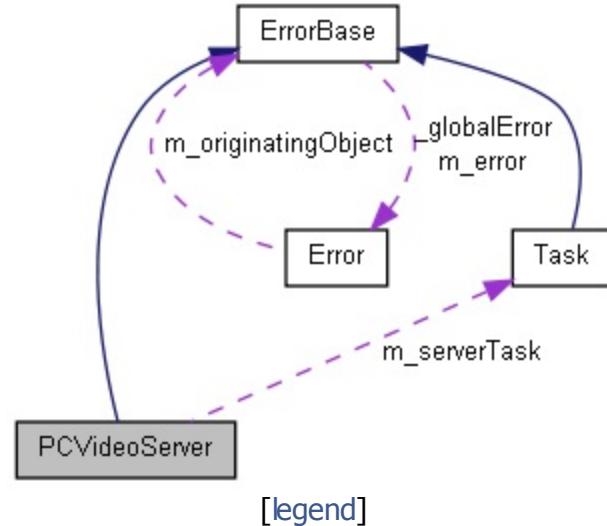
Class the serves images to the PC. [More...](#)

```
#include <PCVideoServer.h>
```

Inheritance diagram for PCVideoServer:



Collaboration diagram for PCVideoServer:



List of all members.

# Public Member Functions

---

**PCVideoServer ()**

Constructor.

---

**~PCVideoServer ()**

Destructor.

---

**unsigned int Release ()**

**void Start ()**

Start sending images to the PC.

---

**void Stop ()**

Stop sending images to the PC.

---

## Detailed Description

Class the serves images to the PC.

---

# Constructor & Destructor Documentation

## **PCVideoServer::~PCVideoServer( )**

Destructor.

Stop serving images and destroy this class.

---

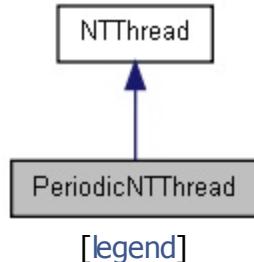
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/Vision/**PCVideoServer.h**
- C:/WindRiver/workspace/WPILib/Vision/PCVideoServer.cpp

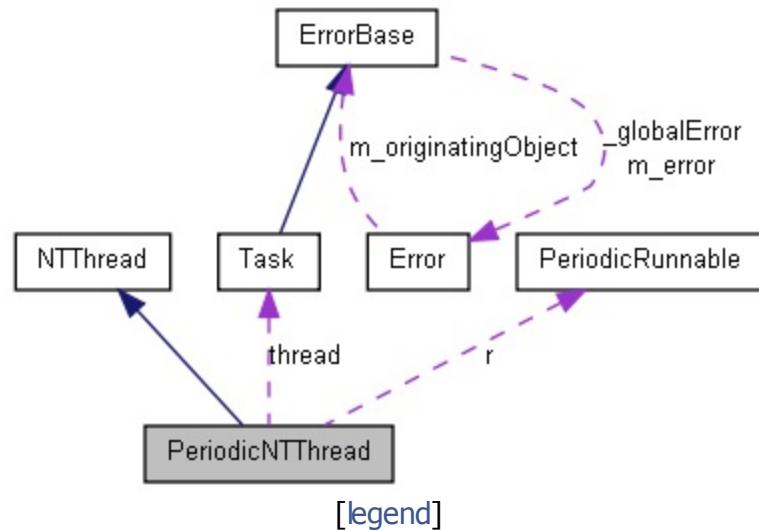


# PeriodicNTThread Class Reference

Inheritance diagram for PeriodicNTThread:



Collaboration diagram for PeriodicNTThread:



List of all members.

## Public Member Functions

**PeriodicNTThread** (**PeriodicRunnable** \*r, const char \*name)

virtual void **stop** ()

stop the thread

virtual bool **isRunning** ()

# Member Function Documentation

**bool PeriodicNTThread::isRunning ( ) [virtual]**

## Returns:

true if the thread is running

Implements **NTThread**.

---

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

- C:/WindRiver/workspace/WPILib/networktables2/thread/**DefaultThreadManager.h**
- C:/WindRiver/workspace/WPILib/networktables2/thread/DefaultThreadManger.cpp

Generated by  1.7.2

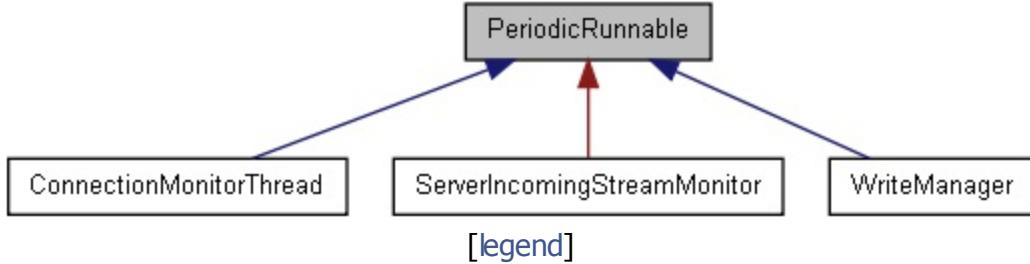


# PeriodicRunnable Class Reference

A runnable where the run method will be called periodically. [More...](#)

```
#include <PeriodicRunnable.h>
```

Inheritance diagram for PeriodicRunnable:



[List of all members.](#)

## Public Member Functions

virtual void **run** ()=0

the method that will be called periodically on a thread

---

## Detailed Description

A runnable where the run method will be called periodically.

### Author:

Mitchell

---

# Member Function Documentation

**virtual void PeriodicRunnable::run( ) [pure virtual]**

the method that will be called periodically on a thread

## Exceptions:

thrown when the thread is supposed to be interrupted and **InterruptedException** stop (implementers should always let this exception fall through)

Implemented in **ConnectionMonitorThread**, **ServerIncomingStreamMonitor**, and **WriteManager**.

---

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

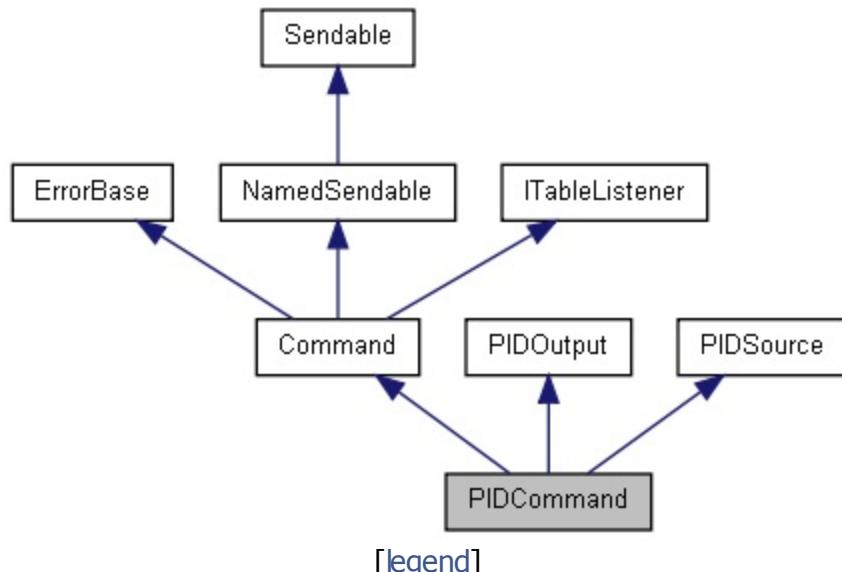
- C:/WindRiver/workspace/WPILib/networktables2/thread/**PeriodicRunnable.h**

[Class List](#)[Class Hierarchy](#)[Class Members](#)

[Public Member Functions](#) |  
[Protected Member Functions](#)

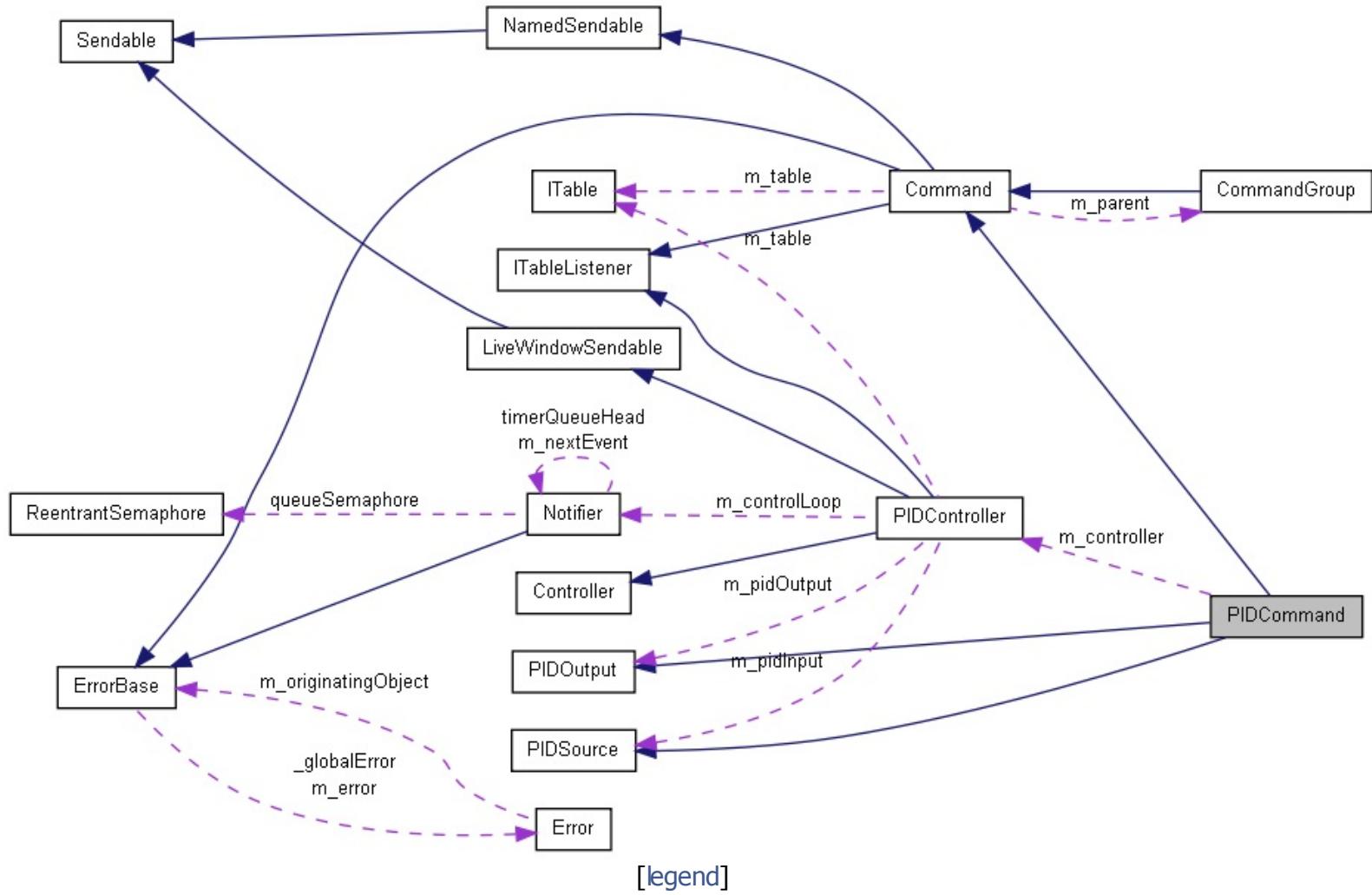
# PIDCommand Class Reference

Inheritance diagram for PIDCommand:



[legend]

Collaboration diagram for PIDCommand:



[legend]

List of all members.

# Public Member Functions

**PIDCommand** (const char \*name, double p, double i, double d)

**PIDCommand** (const char \*name, double p, double i, double d,  
double f, double period)  
**PIDCommand** (const char \*name, double p, double i, double d,  
double f, double period)

**PIDCommand** (double p, double i, double d)

**PIDCommand** (double p, double i, double d, double period)

**PIDCommand** (double p, double i, double d, double f, double  
period)

void **SetSetpointRelative** (double deltaSetpoint)

virtual void **PIDWrite** (float output)

virtual double **PIDGet** ()

virtual void **InitTable** (**ITable** \*table)

Initializes a table for this sendable object.

virtual std::string **GetSmartDashboardType** ()

**PIDController** \* **GetPIDController** ()

virtual void **\_Initialize** ()

virtual void **\_Interrupted** ()

virtual void **\_End** ()

```
void SetSetpoint (double setpoint)
double GetSetpoint ()
double GetPosition ()
virtual double ReturnPIDInput ()=0
virtual void UsePIDOutput (double output)=0
```

---

# Member Function Documentation

**std::string PIDCommand::GetSmartDashboardType( ) [virtual]**

## Returns:

the string representation of the named data type that will be used by the smart dashboard for this sendable

Reimplemented from **Command**.

**void PIDCommand::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

## Parameters:

**subtable** The table to put the values in.

Reimplemented from **Command**.

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

- C:/WindRiver/workspace/WPILib/Commands/**PIDCommand.h**
- C:/WindRiver/workspace/WPILib/Commands/PIDCommand.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

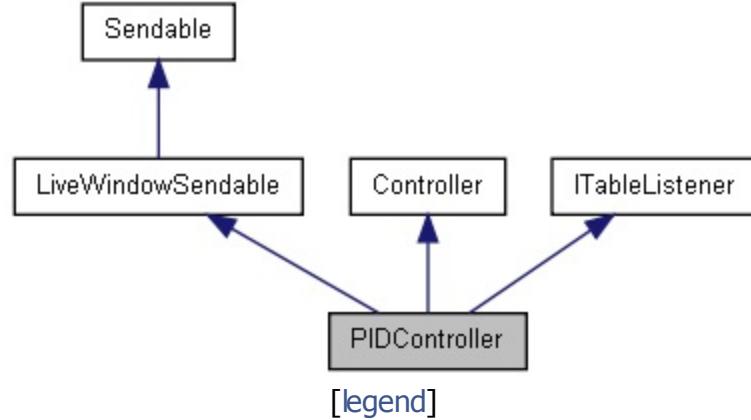
[Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#)

# PIDController Class Reference

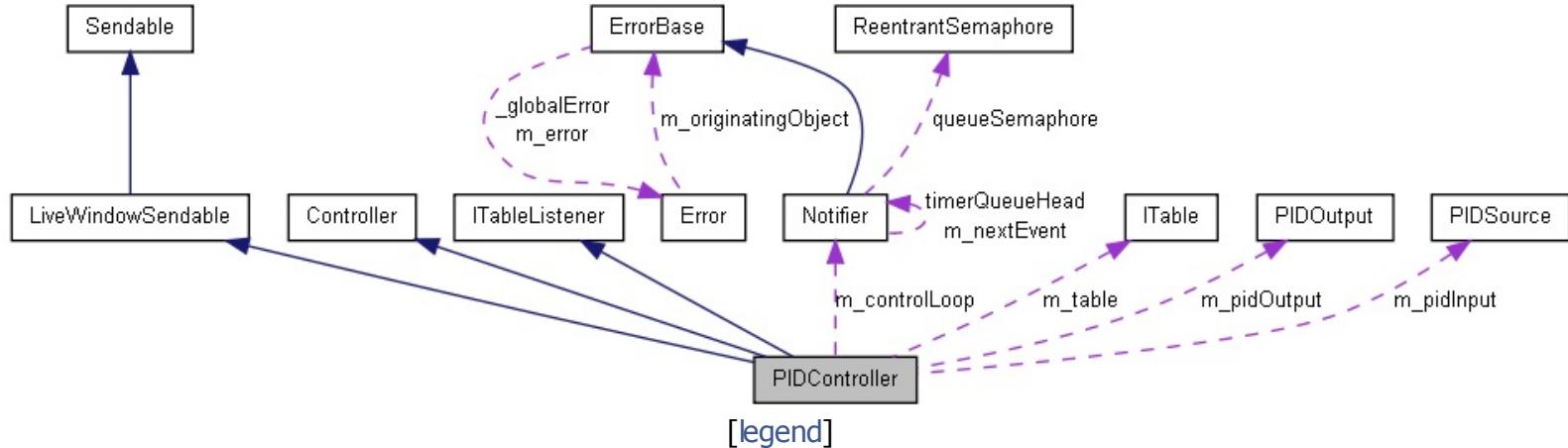
Class implements a PID Control Loop. [More...](#)

```
#include <PIDController.h>
```

Inheritance diagram for PIDController:



Collaboration diagram for PIDController:



List of all members.

## Public Member Functions

**PIDController** (float p, float i, float d, **PIDSource** \*source, **PIDOutput** \*output, float period=0.05)

Allocate a PID object with the given constants for P, I, D.

**PIDController** (float p, float i, float d, float f, **PIDSource** \*source, **PIDOutput** \*output, float period=0.05)

Allocate a PID object with the given constants for P, I, D.

virtual **~PIDController** ()

Free the PID object.

virtual float **Get** ()

Return the current PID result This is always centered on zero and constrained the the max and min outs.

virtual void **SetContinuous** (bool continuous=true)

Set the PID controller to consider the input to be continuous, Rather then using the max and min in as constraints, it considers them to be the same point and automatically calculates the shortest route to the setpoint.

virtual void **SetInputRange** (float minimumInput, float maximumInput)

Sets the maximum and minimum values expected from the input.

virtual void **SetOutputRange** (float mimimumOutput, float maximumOutput)

Sets the minimum and maximum values to write.

virtual void **SetPID** (float p, float i, float d)

Set the PID **Controller** gain parameters.

virtual void **SetPID** (float p, float i, float d, float f)

Set the PID **Controller** gain parameters.

virtual float **GetP** ()

Get the Proportional coefficient.

virtual float **GetI** ()

Get the Integral coefficient.

virtual float **GetD** ()

Get the Differential coefficient.

virtual float **GetF** ()

Get the Feed forward coefficient.

virtual void **SetSetpoint** (float setpoint)

Set the setpoint for the **PIDController**.

virtual float **GetSetpoint** ()

Returns the current setpoint of the **PIDController**.

virtual float **GetError** ()

Retruns the current difference of the input from the setpoint.

virtual void	<b>SetTolerance</b> (float percent)
virtual void	<b>SetAbsoluteTolerance</b> (float absValue)
virtual void	<b>SetPercentTolerance</b> (float percentValue)
virtual bool	<b>OnTarget</b> ()
virtual void	<b>Enable</b> () Begin running the <b>PIDController</b> .
virtual void	<b>Disable</b> () Stop running the <b>PIDController</b> , this sets the output to zero before stopping.
virtual bool	<b>IsEnabled</b> ()  Return true if <b>PIDController</b> is enabled.
virtual void	<b>Reset</b> () Reset the previous error,, the integral term, and disable the controller.
virtual void	<b>InitTable</b> ( <b>ITable</b> *table) Initializes a table for this sendable object.

# Protected Member Functions

---

**DISALLOW\_COPY\_AND\_ASSIGN** ([PIDController](#))

## Protected Attributes

**ITable \* m\_table**

---

## Detailed Description

Class implements a PID Control Loop.

Creates a separate thread which reads the given **PIDSource** and takes care of the integral calculations, as well as writing the given **PIDOutput**

---

# Constructor & Destructor Documentation

```
PIDController::PIDController ( float          Kp,  
                               float          Ki,  
                               float          Kd,  
                               PIDSource *   source,  
                               PIDOutput *  output,  
                               float         period = 0.05  
                           )
```

Allocate a PID object with the given constants for P, I, D.

## Parameters:

- Kp** the proportional coefficient
- Ki** the integral coefficient
- Kd** the derivative coefficient
- source** The **PIDSource** object that is used to get values
- output** The **PIDOutput** object that is set to the output value
- period** the loop time for doing calculations. This particularly effects calculations of the integral and differential terms. The default is 50ms.

```
PIDController::PIDController ( float          Kp,  
                               float          Ki,  
                               float          Kd,  
                               float          Kf,  
                               PIDSource *   source,  
                               PIDOutput *  output,  
                               float         period = 0.05  
                           )
```

Allocate a PID object with the given constants for P, I, D.

## Parameters:

- Kp** the proportional coefficient
- Ki** the integral coefficient
- Kd** the derivative coefficient
- source** The **PIDSource** object that is used to get values
- output** The **PIDOutput** object that is set to the output value

**period** the loop time for doing calculations. This particularly effects calculations of the integral and differential terms. The default is 50ms.

# Member Function Documentation

## **float PIDController::Get( ) [virtual]**

Return the current PID result This is always centered on zero and constrained the the max and min outs.

### **Returns:**

the latest calculated output

## **float PIDController::GetD( ) [virtual]**

Get the Differential coefficient.

### **Returns:**

differential coefficient

## **float PIDController::GetError( ) [virtual]**

Retruns the current difference of the input from the setpoint.

### **Returns:**

the current error

## **float PIDController::GetF( ) [virtual]**

Get the Feed forward coefficient.

### **Returns:**

Feed forward coefficient

## **float PIDController::GetI( ) [virtual]**

Get the Integral coefficient.

### **Returns:**

integral coefficient

**float PIDController::GetP( )** [virtual]

Get the Proportional coefficient.

**Returns:**

proportional coefficient

**float PIDController::GetSetpoint( )** [virtual]

Returns the current setpoint of the **PIDController**.

**Returns:**

the current setpoint

**void PIDController::InitTable( ITable \* subtable )** [virtual]

Initializes a table for this sendable object.

**Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

**void PIDController::SetContinuous( bool continuous = true )** [virtual]

Set the PID controller to consider the input to be continuous, Rather then using the max and min in as constraints, it considers them to be the same point and automatically calculates the shortest route to the setpoint.

**Parameters:**

**continuous** Set to true turns on continuous, false turns off continuous

**void PIDController::SetInputRange( float minimumInput,  
                              float maximumInput  
                              )** [virtual]

Sets the maximum and minimum values expected from the input.

**Parameters:**

**minimumInput** the minimum value expected from the input  
**maximumInput** the maximum value expected from the output

```
void PIDController::SetOutputRange ( float minimumOutput,
                                    float maximumOutput
                                    ) [virtual]
```

Sets the minimum and maximum values to write.

#### Parameters:

**minimumOutput** the minimum value to write to the output  
**maximumOutput** the maximum value to write to the output

```
void PIDController::SetPID ( float p,
                            float i,
                            float d
                            ) [virtual]
```

Set the PID **Controller** gain parameters.

Set the proportional, integral, and differential coefficients.

#### Parameters:

**p** Proportional coefficient  
**i** Integral coefficient  
**d** Differential coefficient

```
void PIDController::SetPID ( float p,
                            float i,
                            float d,
                            float f
                            ) [virtual]
```

Set the PID **Controller** gain parameters.

Set the proportional, integral, and differential coefficients.

#### Parameters:

- p** Proportional coefficient
- i** Integral coefficient
- d** Differential coefficient
- f** Feed forward coefficient

## **void PIDController::SetSetpoint( float setpoint ) [virtual]**

Set the setpoint for the **PIDController**.

### **Parameters:**

**setpoint** the desired setpoint

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

- C:/WindRiver/workspace/WPILib/**PIDController.h**
- C:/WindRiver/workspace/WPILib/PIDController.cpp

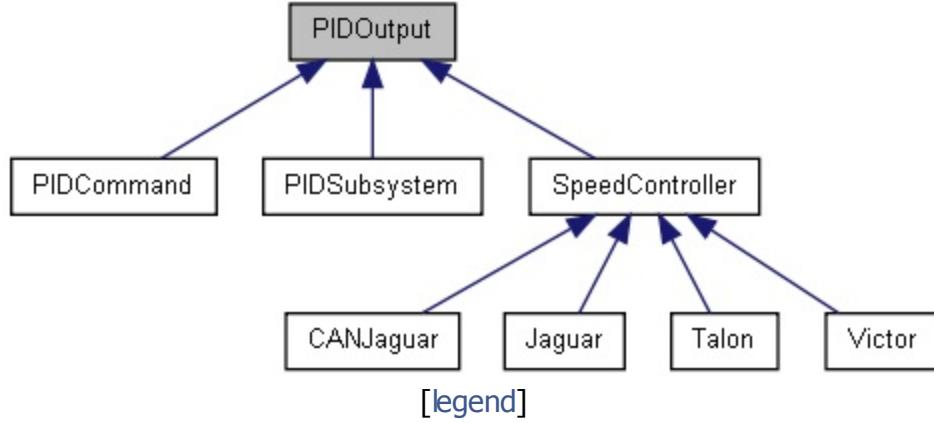


# PIDOutput Class Reference

**PIDOutput** interface is a generic output for the PID class. [More...](#)

```
#include <PIDOutput.h>
```

Inheritance diagram for PIDOutput:



[List of all members.](#)

## Public Member Functions

virtual void **PIDWrite** (float output)=0

---

## Detailed Description

**PIDOutput** interface is a generic output for the PID class.

PWMs use this class. Users implement this interface to allow for a **PIDController** to read directly from the inputs

---

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

- C:/WindRiver/workspace/WPILib/**PIDOutput.h**
- 

Generated by  1.7.2

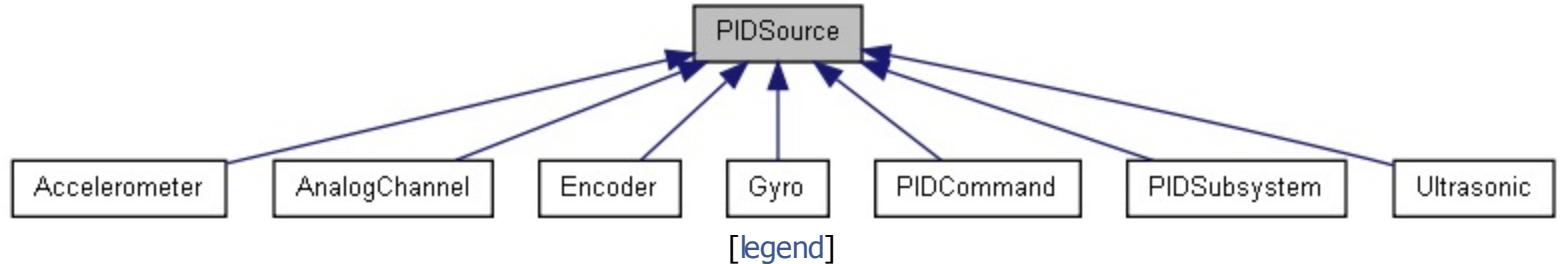


# PIDSource Class Reference

**PIDSource** interface is a generic sensor source for the PID class. [More...](#)

```
#include <PIDSource.h>
```

Inheritance diagram for PIDSource:



List of all members.

## Public Member Functions

virtual double **PIDGet ()=0**

---

## Detailed Description

**PIDSource** interface is a generic sensor source for the PID class.

All sensors that can be used with the PID class will implement the **PIDSource** that returns a standard value that will be used in the PID code.

---

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

- C:/WindRiver/workspace/WPILib/**PIDSource.h**

Generated by  1.7.2

[Class List](#)[Class Hierarchy](#)[Class Members](#)

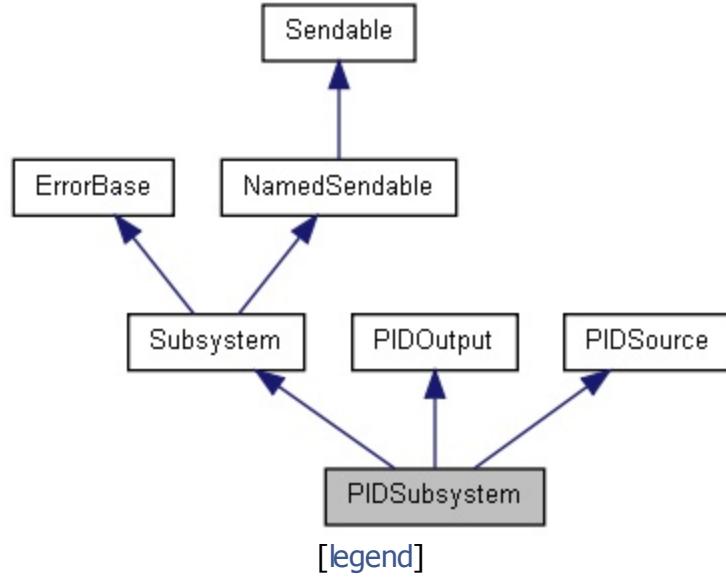
[Public Member Functions](#) |  
[Protected Member Functions](#)

# PIDSubsystem Class Reference

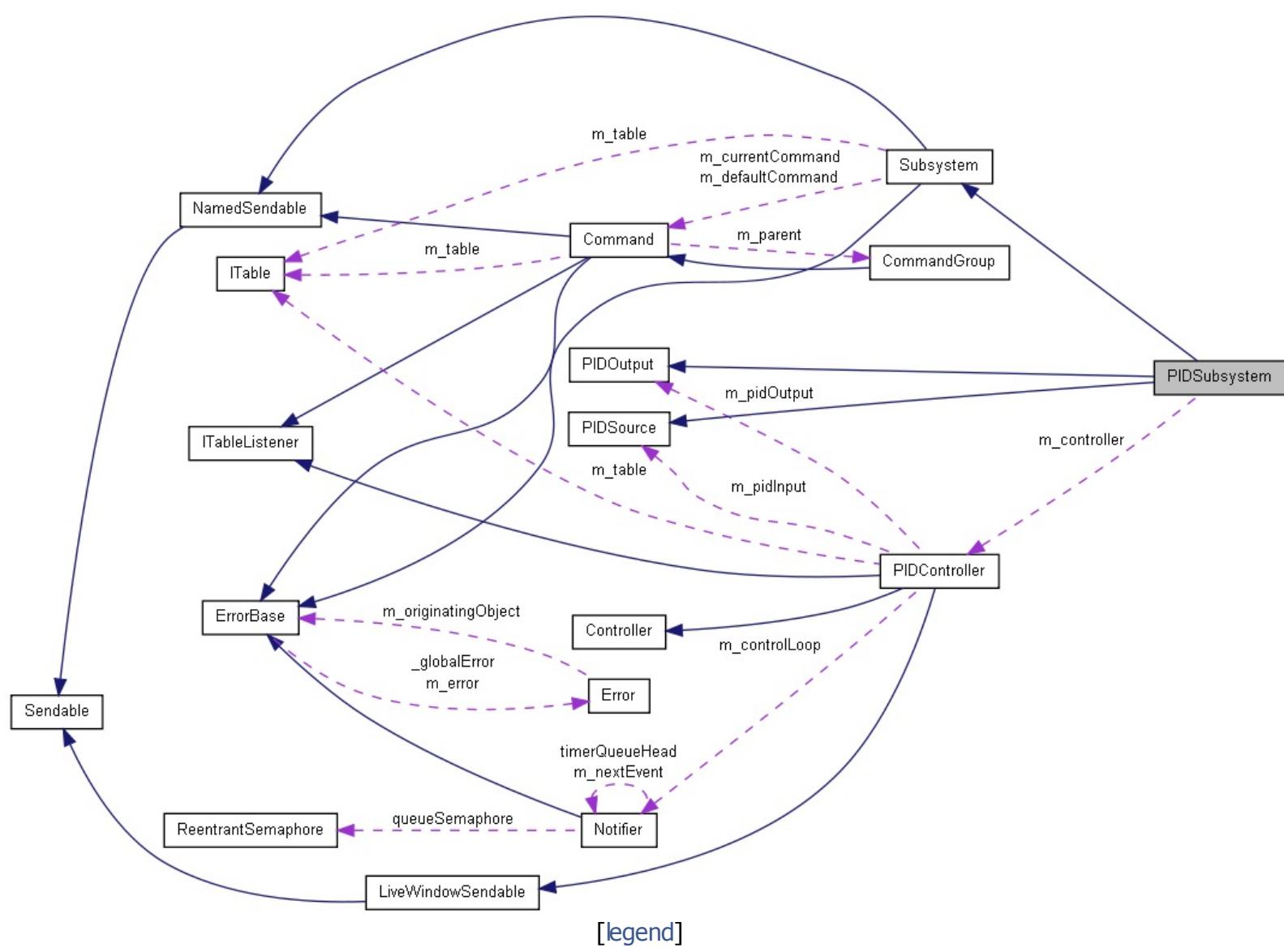
This class is designed to handle the case where there is a **Subsystem** which uses a single **PIDController** almost constantly (for instance, an elevator which attempts to stay at a constant height). [More...](#)

```
#include <PIDSubsystem.h>
```

Inheritance diagram for PIDSubsystem:



Collaboration diagram for PIDSubsystem:



List of all members.

**PIDSubsystem** (const char \*name, double p, double i, double d)  
 Instantiates a **PIDSubsystem** that will use the given p, i and d values.

**PIDSubsystem** (const char \*name, double p, double i, double d, double f)

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

**PIDSubsystem** (const char \*name, double p, double i, double d, double f, double period)

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

**PIDSubsystem** (double p, double i, double d)

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

**PIDSubsystem** (double p, double i, double d, double f)

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

**PIDSubsystem** (double p, double i, double d, double f, double period)

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

void **Enable** ()

Enables the internal **PIDController**.

void **Disable** ()

Disables the internal **PIDController**.

virtual void **PIDWrite** (float output)

virtual double **PIDGet** ()

void **SetSetpoint** (double setpoint)

Sets the setpoint to the given value.

void **SetSetpointRelative** (double deltaSetpoint)

Adds the given value to the setpoint.

void **SetInputRange** (float minimumInput, float maximumInput)

Sets the maximum and minimum values expected from the input.

double **GetSetpoint** ()

Return the current setpoint.

double **GetPosition** ()

Returns the current position.

virtual void **SetAbsoluteTolerance** (float absValue)

virtual void **SetPercentTolerance** (float percent)

virtual bool **OnTarget** ()

virtual void **InitTable** (ITable \*table)

Initializes a table for this sendable object.

virtual std::string **GetSmartDashboardType** ()

## Protected Member Functions

**PIDController \*** **GetPIDController ()**

Returns the **PIDController** used by this **PIDSubsystem**.

**virtual double** **ReturnPIDInput ()=0**

**virtual void** **UsePIDOutput (double output)=0**

## Detailed Description

This class is designed to handle the case where there is a **Subsystem** which uses a single **PIDController** almost constantly (for instance, an elevator which attempts to stay at a constant height).

It provides some convenience methods to run an internal **PIDController**. It also allows access to the internal **PIDController** in order to give total control to the programmer.

---

# Constructor & Destructor Documentation

```
PIDSubsystem::PIDSubsystem ( const char * name,  
                           double      p,  
                           double      i,  
                           double      d  
                         )
```

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

## Parameters:

**name** the name  
**p** the proportional value  
**i** the integral value  
**d** the derivative value

```
PIDSubsystem::PIDSubsystem ( const char * name,  
                           double      p,  
                           double      i,  
                           double      d,  
                           double      f  
                         )
```

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

## Parameters:

**name** the name  
**p** the proportional value  
**i** the integral value  
**d** the derivative value  
**f** the feedforward value

```
PIDSubsystem::PIDSubsystem ( const char * name,  
                           double      p,  
                           double      i,  
                           double      d,  
                           double      f,
```

```
    double period  
)
```

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

It will also space the time between PID loop calculations to be equal to the given period.

#### Parameters:

- name** the name
- p** the proportional value
- i** the integral value
- d** the derivative value
- f** the feedforward value
- period** the time (in seconds) between calculations

```
PIDSubsystem::PIDSubsystem ( double p,  
                            double i,  
                            double d  
)
```

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

It will use the class name as its name.

#### Parameters:

- p** the proportional value
- i** the integral value
- d** the derivative value

```
PIDSubsystem::PIDSubsystem ( double p,  
                            double i,  
                            double d,  
                            double f  
)
```

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

It will use the class name as its name.

### Parameters:

- p** the proportional value
- i** the integral value
- d** the derivative value
- f** the feedforward value

---

**PIDSubsystem::PIDSubsystem ( double p,  
double i,  
double d,  
double f,  
double period  
)**

Instantiates a **PIDSubsystem** that will use the given p, i and d values.

It will use the class name as its name. It will also space the time between PID loop calculations to be equal to the given period.

### Parameters:

- p** the proportional value
- i** the integral value
- d** the derivative value
- f** the feedforward value
- period** the time (in seconds) between calculations

# Member Function Documentation

**PIDController \* PIDSubsystem::GetPIDController( )** [protected]

Returns the **PIDController** used by this **PIDSubsystem**.

Use this if you would like to fine tune the pid loop.

**Returns:**

the **PIDController** used by this **PIDSubsystem**

**double PIDSubsystem::GetPosition( )**

Returns the current position.

**Returns:**

the current position

**double PIDSubsystem::GetSetpoint( )**

Return the current setpoint.

**Returns:**

The current setpoint

**std::string PIDSubsystem::GetSmartDashboardType( )** [virtual]

**Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Reimplemented from **Subsystem**.

**void PIDSubsystem::InitTable( ITable \* subtable )** [virtual]

Initializes a table for this sendable object.

**Parameters:**

**subtable** The table to put the values in.

Reimplemented from **Subsystem**.

```
void PIDSubsystem::SetInputRange( float minimumInput,
                           float maximumInput
)
```

Sets the maximum and minimum values expected from the input.

**Parameters:**

**minimumInput** the minimum value expected from the input

**maximumInput** the maximum value expected from the output

```
void PIDSubsystem::SetSetpoint( double setpoint )
```

Sets the setpoint to the given value.

If **SetRange(...)** was called, then the given setpoint will be trimmed to fit within the range.

**Parameters:**

**setpoint** the new setpoint

```
void PIDSubsystem::SetSetpointRelative( double deltaSetpoint )
```

Adds the given value to the setpoint.

If **SetRange(...)** was used, then the bounds will still be honored by this method.

**Parameters:**

**deltaSetpoint** the change in the setpoint

---

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

- C:/WindRiver/workspace/WPILib/Commands/**PIDSubsystem.h**
- C:/WindRiver/workspace/WPILib/Commands/PIDSubsystem.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)[Kinect](#) › [Point4](#) ›

Public Attributes

# Kinect::Point4 Struct Reference

---

List of all members.

## Public Attributes

float <b>x</b>
float <b>y</b>
float <b>z</b>
float <b>w</b>

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

- C:/WindRiver/workspace/WPILib/**Kinect.h**

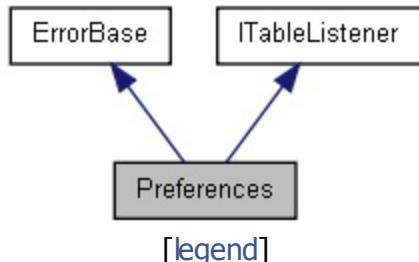
[Public Member Functions](#) |  
[Static Public Member Functions](#)

# Preferences Class Reference

The preferences class provides a relatively simple way to save important values to the cRIO to access the next time the cRIO is booted. More...

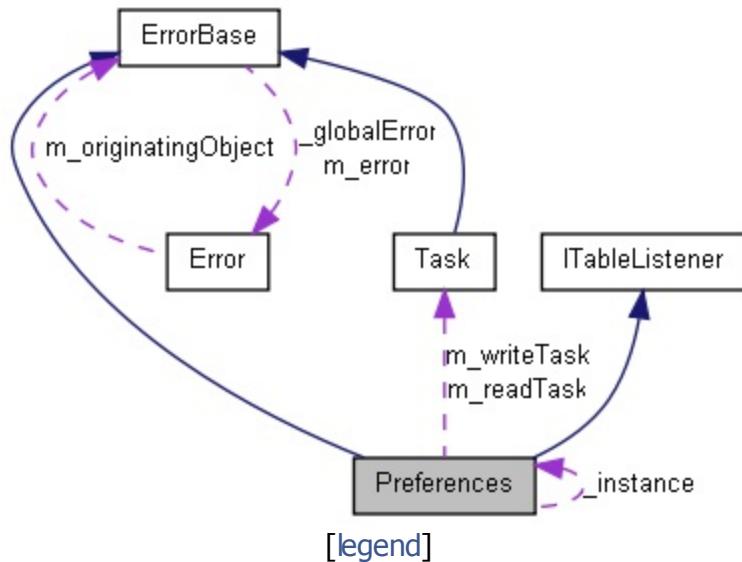
```
#include <Preferences.h>
```

Inheritance diagram for Preferences:



[legend]

Collaboration diagram for Preferences:



[legend]

List of all members.

# Public Member Functions

std::vector< std::string >	<b>GetKeys ()</b>	Returns a vector of all the keys.
std::string	<b>GetString</b> (const char *key, const char *defaultValue="")	Returns the string at the given key.
int	<b>GetString</b> (const char *key, char *value, int valueSize, const char *defaultValue="")	Returns the string at the given key.
int	<b>GetInt</b> (const char *key, int defaultValue=0)	Returns the int at the given key.
double	<b>GetDouble</b> (const char *key, double defaultValue=0.0)	Returns the double at the given key.
float	<b>GetFloat</b> (const char *key, float defaultValue=0.0)	Returns the float at the given key.
bool	<b>GetBoolean</b> (const char *key, bool defaultValue=false)	Returns the boolean at the given key.
INT64	<b>GetLong</b> (const char *key, INT64 defaultValue=0)	Returns the long (INT64) at the given key.
void	<b>PutString</b> (const char *key, const char *value)	Puts the given string into the preferences table.
void	<b>PutInt</b> (const char *key, int value)	Puts the given int into the preferences table.
void	<b>PutDouble</b> (const char *key, double value)	Puts the given double into the preferences table.
void	<b>PutFloat</b> (const char *key, float value)	Puts the given float into the preferences table.
void	<b>PutBoolean</b> (const char *key, bool value)	Puts the given boolean into the preferences table.
void	<b>PutLong</b> (const char *key, INT64 value)	Puts the given long (INT64) into the preferences table.
void	<b>Save ()</b>	Saves the preferences to a file on the cRIO.
bool	<b>ContainsKey</b> (const char *key)	Returns whether or not there is a key with the given name.
void	<b>Remove</b> (const char *key)	Remove a preference.

```
void ValueChanged (ITable *source, const std::string &key,  
EntryValue value, bool isNew)
```

Called when a key-value pair is changed in a **ITable**  
WARNING: If a new key-value is put in this method value  
changed will immediatly be called which could lead to  
recursive code.

---

```
static Preferences * GetInstance ()
```

Get the one and only **Preferences** object.

---

## Detailed Description

The preferences class provides a relatively simple way to save important values to the cRIO to access the next time the cRIO is booted.

This class loads and saves from a file inside the cRIO. The user can not access the file directly, but may modify values at specific fields which will then be saved to the file when **Save()** is called.

This class is thread safe.

This will also interact with **NetworkTable** by creating a table called "Preferences" with all the key-value pairs. To save using **NetworkTable**, simply set the boolean at position "`~S A V E~`" to true. Also, if the value of any variable is "" in the **NetworkTable**, then that represents non-existence in the **Preferences** table

---

# Member Function Documentation

## **bool Preferences::ContainsKey ( const char \* key )**

Returns whether or not there is a key with the given name.

### Parameters:

**key** the key

### Returns:

if there is a value at the given key

## **bool Preferences::GetBoolean ( const char \* key,                                   bool              defaultValue = false                                   )**

Returns the boolean at the given key.

If this table does not have a value for that position, then the given defaultValue value will be returned.

### Parameters:

**key** the key

**defaultValue** the value to return if none exists in the table

### Returns:

either the value in the table, or the defaultValue

## **double Preferences::GetDouble ( const char \* key,                                   double          defaultValue = 0.0                                   )**

Returns the double at the given key.

If this table does not have a value for that position, then the given defaultValue value will be returned.

### Parameters:

**key** the key

**defaultValue** the value to return if none exists in the table

**Returns:**

either the value in the table, or the defaultValue

```
float Preferences::GetFloat( const char * key,  
                           float           defaultValue = 0.0  
                         )
```

Returns the float at the given key.

If this table does not have a value for that position, then the given defaultValue value will be returned.

**Parameters:**

**key** the key

**defaultValue** the value to return if none exists in the table

**Returns:**

either the value in the table, or the defaultValue

**Preferences** \* Preferences::GetInstance( ) [static]

Get the one and only **Preferences** object.

**Returns:**

pointer to the **Preferences**

```
int Preferences::GetInt( const char * key,  
                        int           defaultValue = 0  
                      )
```

Returns the int at the given key.

If this table does not have a value for that position, then the given defaultValue value will be returned.

**Parameters:**

**key** the key

**defaultValue** the value to return if none exists in the table

**Returns:**

either the value in the table, or the defaultValue

**std::vector< std::string > Preferences::GetKeys( )**

Returns a vector of all the keys.

**Returns:**

a vector of the keys

**INT64 Preferences::GetLong ( const char \* key,  
                                  INT64              defaultValue = 0  
                                  )**

Returns the long (INT64) at the given key.

If this table does not have a value for that position, then the given defaultValue value will be returned.

**Parameters:**

**key**                 the key

**defaultValue** the value to return if none exists in the table

**Returns:**

either the value in the table, or the defaultValue

**int Preferences::GetString ( const char \* key,  
                                  char \*            value,  
                                  int                valueSize,  
                                  const char \*  defaultValue = ""  
                                  )**

Returns the string at the given key.

If this table does not have a value for that position, then the given defaultValue will be returned.

**Parameters:**

**key**                 the key

**value** the buffer to copy the value into

**valueSize** the size of value

**defaultValue** the value to return if none exists in the table

### Returns:

The size of the returned string

```
std::string Preferences::GetString ( const char * key,  
                                    const char * defaultValue = "" )
```

Returns the string at the given key.

If this table does not have a value for that position, then the given defaultValue will be returned.

### Parameters:

**key** the key

**defaultValue** the value to return if none exists in the table

### Returns:

either the value in the table, or the defaultValue

```
void Preferences::PutBoolean ( const char * key,  
                             bool value )
```

Puts the given boolean into the preferences table.

The key may not have any whitespace nor an equals sign

This will **NOT** save the value to memory between power cycles, to do that you must call **Save()** (which must be used with care) at some point after calling this.

### Parameters:

**key** the key

**value** the value

```
void Preferences::PutDouble ( const char * key,
```

```
    double      value  
    )
```

Puts the given double into the preferences table.

The key may not have any whitespace nor an equals sign

This will **NOT** save the value to memory between power cycles, to do that you must call [Save\(\)](#) (which must be used with care) at some point after calling this.

#### Parameters:

**key** the key  
**value** the value

```
void Preferences::PutFloat( const char * key,  
                           float       value  
                           )
```

Puts the given float into the preferences table.

The key may not have any whitespace nor an equals sign

This will **NOT** save the value to memory between power cycles, to do that you must call [Save\(\)](#) (which must be used with care) at some point after calling this.

#### Parameters:

**key** the key  
**value** the value

```
void Preferences::PutInt( const char * key,  
                          int        value  
                          )
```

Puts the given int into the preferences table.

The key may not have any whitespace nor an equals sign

This will **NOT** save the value to memory between power cycles, to do that you must call [Save\(\)](#) (which must be used with care) at some point after calling this.

## Parameters:

**key** the key  
**value** the value

```
void Preferences::PutLong ( const char * key,  
                           INT64      value  
                         )
```

Puts the given long (INT64) into the preferences table.

The key may not have any whitespace nor an equals sign

This will **NOT** save the value to memory between power cycles, to do that you must call [Save\(\)](#) (which must be used with care) at some point after calling this.

## Parameters:

**key** the key  
**value** the value

```
void Preferences::PutString ( const char * key,  
                            const char * value  
                          )
```

Puts the given string into the preferences table.

The value may not have quotation marks, nor may the key have any whitespace nor an equals sign

This will **NOT** save the value to memory between power cycles, to do that you must call [Save\(\)](#) (which must be used with care). at some point after calling this.

## Parameters:

**key** the key  
**value** the value

```
void Preferences::Remove ( const char * key )
```

Remove a preference.

## Parameters:

**key** the key

## void Preferences::Save( )

Saves the preferences to a file on the cRIO.

This should **NOT** be called often. Too many writes can damage the cRIO's flash memory. While it is ok to save once or twice a match, this should never be called every run of **IterativeRobot#TeleopPeriodic()**, etc.

The actual writing of the file is done in a separate thread. However, any call to a get or put method will wait until the table is fully saved before continuing.

```
void Preferences::ValueChanged( ITable * source,  
                               const std::string & key,  
                               EntryValue value,  
                               bool isNew  
) [virtual]
```

Called when a key-value pair is changed in a **ITable** **WARNING:** If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

## Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

---

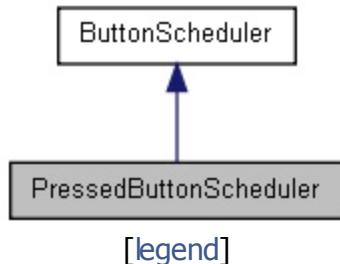
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/**Preferences.h**
- C:/WindRiver/workspace/WPILib/Preferences.cpp

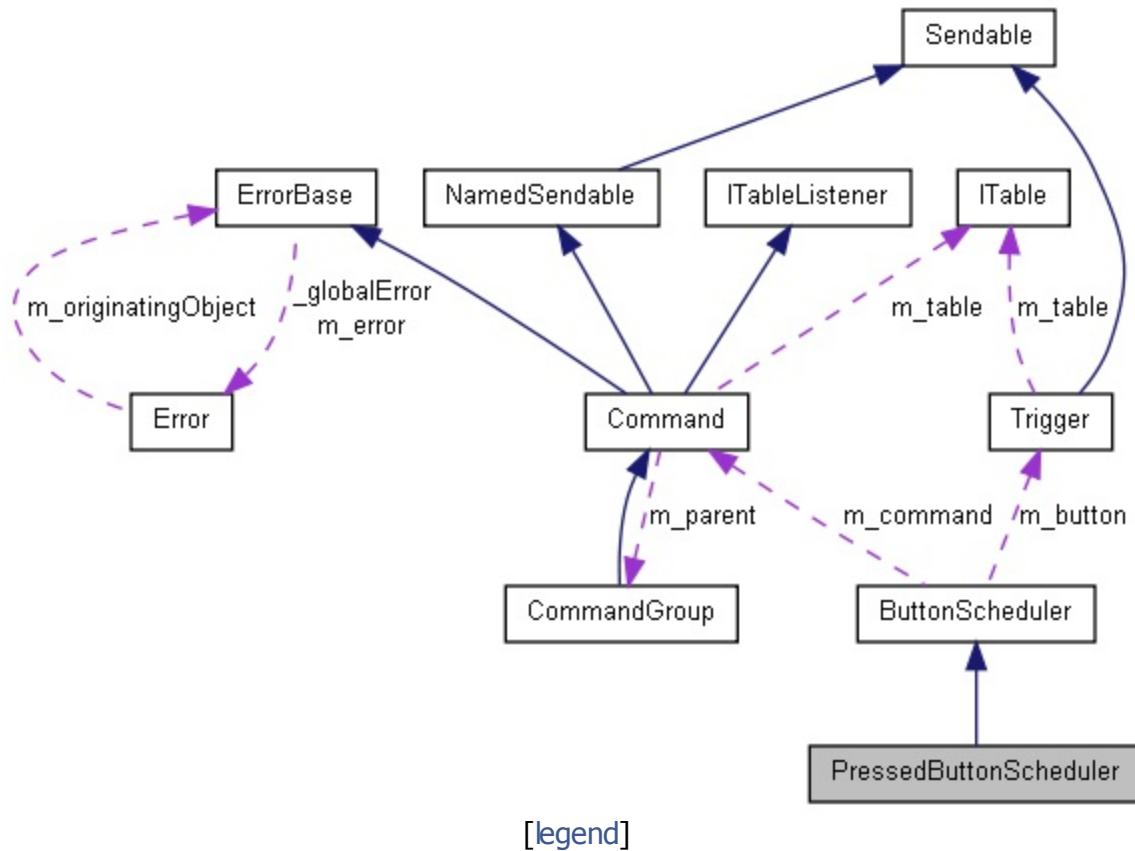


# PressedButtonScheduler Class Reference

Inheritance diagram for PressedButtonScheduler:



Collaboration diagram for PressedButtonScheduler:



List of all members.

# Public Member Functions

**PressedButtonScheduler** (bool last, **Trigger** \*button, **Command** \*orders)

virtual void **Execute** ()

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

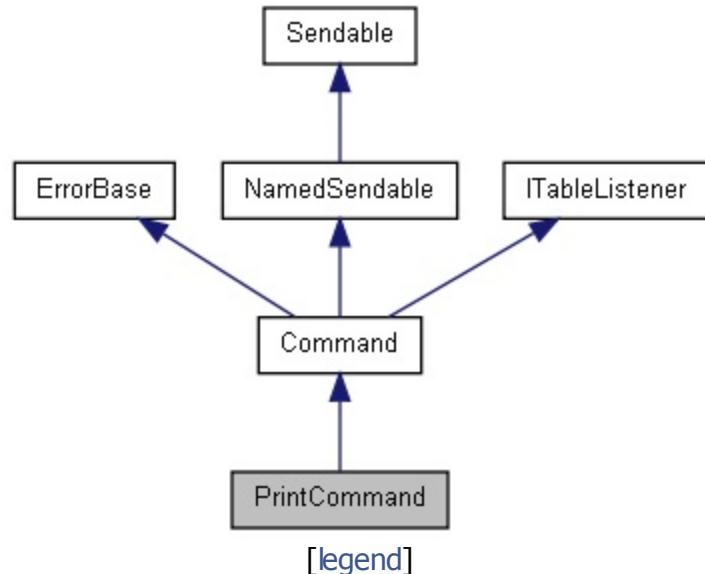
- C:/WindRiver/workspace/WPILib/Buttons/**PressedButtonScheduler.h**
- C:/WindRiver/workspace/WPILib/Buttons/PressedButtonScheduler.cpp

Generated by  1.7.2

[Public Member Functions](#) |  
[Protected Member Functions](#)

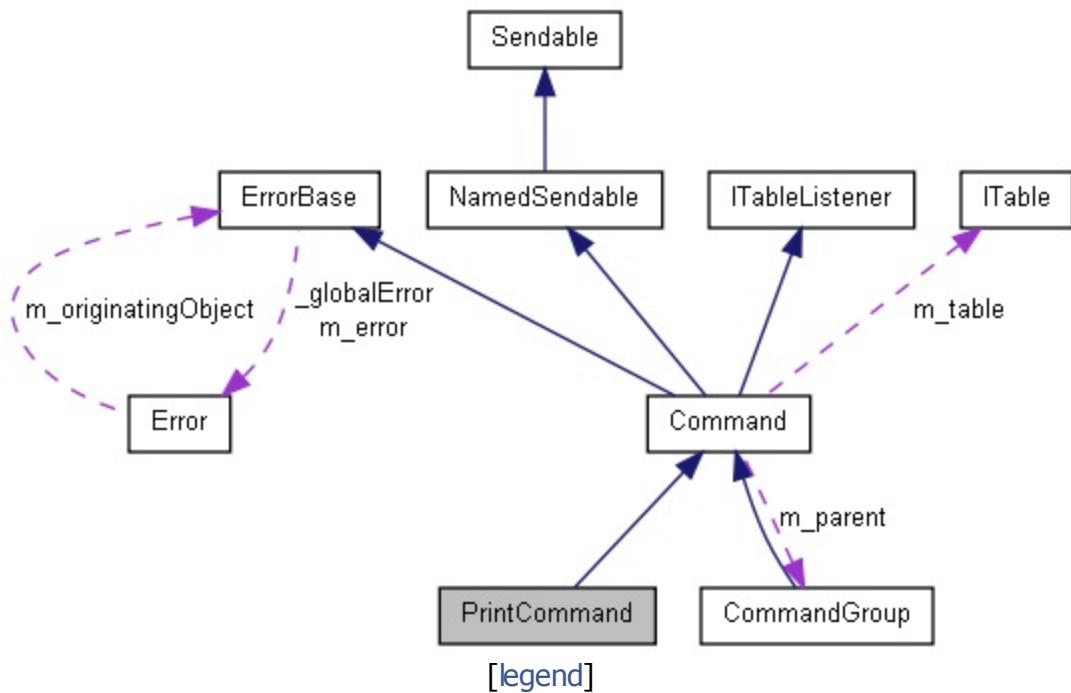
# PrintCommand Class Reference

Inheritance diagram for PrintCommand:



[legend]

Collaboration diagram for PrintCommand:



List of all members.

# Public Member Functions

**PrintCommand** (const char \*message)

virtual void **Initialize** ()

The initialize method is called the first time this **Command** is run after being started.

virtual void **Execute** ()

The execute method is called repeatedly until this **Command** either finishes or is canceled.

virtual bool **IsFinished** ()

Returns whether this command is finished.

virtual void **End** ()

Called when the command ended peacefully.

virtual void **Interrupted** ()

Called when the command ends because somebody called **cancel()** or another command shared the same requirements as this one, and booted it out.

# Member Function Documentation

## **void PrintCommand::End( )** [protected, virtual]

Called when the command ended peacefully.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Implements [Command](#).

## **void PrintCommand::Interrupted( )** [protected, virtual]

Called when the command ends because somebody called [cancel\(\)](#) or another command shared the same requirements as this one, and booted it out.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Generally, it is useful to simply call the [end\(\)](#) method within this method

Implements [Command](#).

## **bool PrintCommand::IsFinished( )** [protected, virtual]

Returns whether this command is finished.

If it is, then the command will be removed and [end\(\)](#) will be called.

It may be useful for a team to reference the [isTimedOut\(\)](#) method for time-sensitive commands.

### **Returns:**

whether this command is finished.

### **See also:**

[Command::isTimedOut\(\)](#) [isTimedOut\(\)](#)

Implements [Command](#).

- C:/WindRiver/workspace/WPILib/Commands/**PrintCommand.h**
  - C:/WindRiver/workspace/WPILib/Commands/PrintCommand.cpp
- 

Generated by [doxygen](#) 1.7.2

[Class List](#)[Class Hierarchy](#)[Class Members](#)

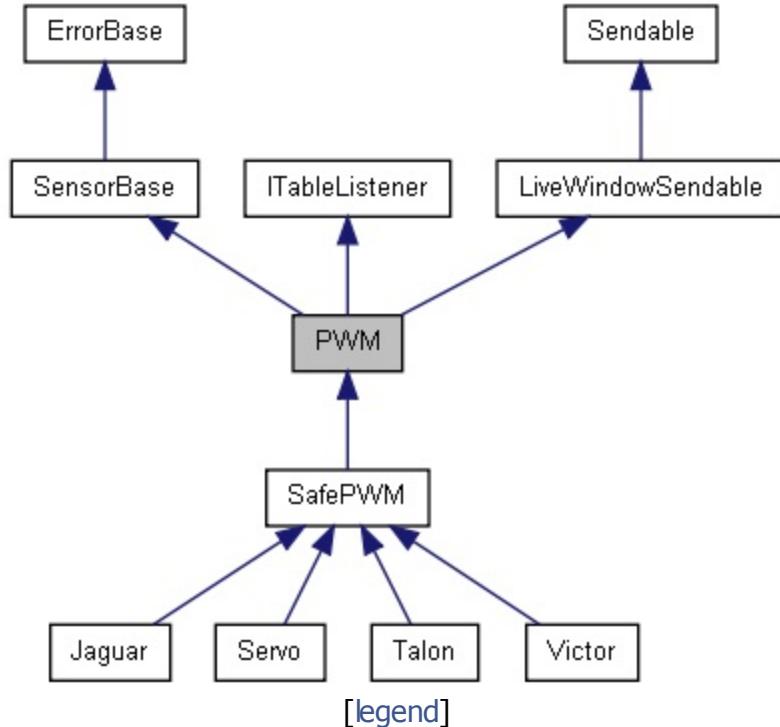
[Public Types](#) | [Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#) |  
[Static Protected Attributes](#) | [Friends](#)

# PWM Class Reference

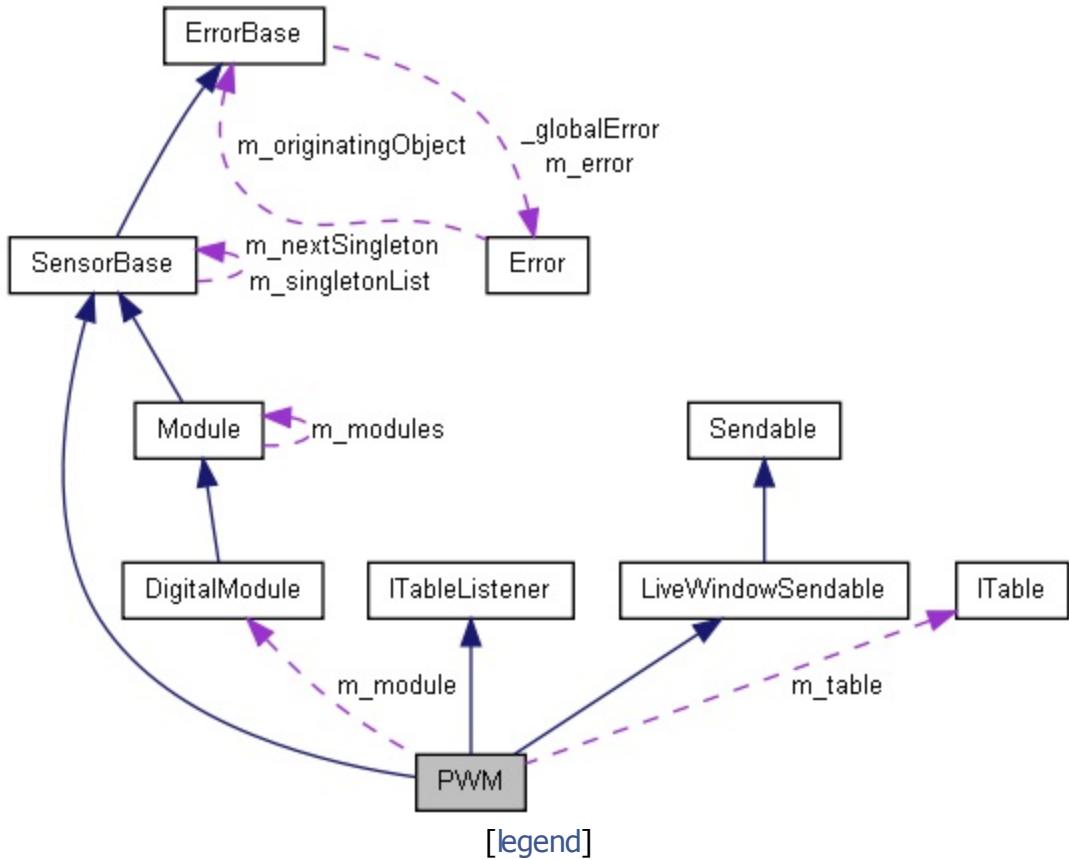
Class implements the **PWM** generation in the FPGA. More...

```
#include <PWM.h>
```

Inheritance diagram for PWM:



Collaboration diagram for PWM:



List of all members.

## Public Types

```
enum PeriodMultiplier { kPeriodMultiplier_1X = 1,  
                      kPeriodMultiplier_2X = 2, kPeriodMultiplier_4X = 4 }
```

**PWM** (UINT32 channel)  
Allocate a **PWM** in the default module given a channel.

**PWM** (UINT8 moduleNumber, UINT32 channel)

Allocate a **PWM** given a module and channel.

virtual **~PWM** ()  
Free the **PWM** channel.

virtual void **SetRaw** (UINT8 value)  
Set the **PWM** value directly to the hardware.

virtual UINT8 **GetRaw** ()  
Get the **PWM** value directly from the hardware.

void **SetPeriodMultiplier** (PeriodMultiplier mult)  
Slow down the **PWM** signal for old devices.

void **EnableDeadbandElimination** (bool eliminateDeadband)  
Optionally eliminate the deadband from a speed controller.

void **SetBounds** (INT32 max, INT32 deadbandMax, INT32 center,  
INT32 deadbandMin, INT32 min)  
Set the bounds on the **PWM** values.

UINT32 **GetChannel** ()

UINT32 **GetModuleNumber** ()

virtual void **SetPosition** (float pos)  
Set the **PWM** value based on a position.

virtual float **GetPosition** ()  
Get the **PWM** value in terms of a position.

virtual void **SetSpeed** (float speed)  
Set the **PWM** value based on a speed.

virtual float **GetSpeed** ()

Get the **PWM** value in terms of speed.

**void ValueChanged (ITable \*source, const std::string &key, EntryValue value, bool isNew)**

Called when a key-value pair is changed in a **ITable** **WARNING:**  
If a new key-value is put in this method value changed will  
immediately be called which could lead to recursive code.

**void UpdateTable ()**

Update the table for this sendable object with the latest values.

**void StartLiveWindowMode ()**

Start having this sendable object automatically respond to value  
changes reflect the value on the table.

**void StopLiveWindowMode ()**

Stop having this sendable object automatically respond to value  
changes.

**std::string GetSmartDashboardType ()**

**void InitTable (ITable \*subTable)**

Initializes a table for this sendable object.

**ITable \* GetTable ()**

## Protected Attributes

bool	<b>m_eliminateDeadband</b>
INT32	<b>m_maxPwm</b>
INT32	<b>m_deadbandMaxPwm</b>
INT32	<b>m_centerPwm</b>
INT32	<b>m_deadbandMinPwm</b>
INT32	<b>m_minPwm</b>
<b>ITable</b> *	<b>m_table</b>

static const UINT32	<b>kDefaultPwmPeriod</b> = 774 kDefaultPwmPeriod is "ticks" where each tick is 6.525us
---------------------	---

static const UINT32	<b>kDefaultMinPwmHigh</b> = 102 kDefaultMinPwmHigh is "ticks" where each tick is 6.525us
---------------------	---

static const INT32	<b>kPwmDisabled</b> = 0
--------------------	-------------------------

## **Friends**

class **DigitalModule**

---

## Detailed Description

Class implements the **PWM** generation in the FPGA.

The values supplied as arguments for **PWM** outputs range from -1.0 to 1.0. They are mapped to the hardware dependent values, in this case 0-255 for the FPGA. Changes are immediately sent to the FPGA, and the update occurs at the next FPGA cycle. There is no delay.

As of revision 0.1.10 of the FPGA, the FPGA interprets the 0-255 values as follows:

- 255 = full "forward"
  - 254 to 129 = linear scaling from "full forward" to "center"
  - 128 = center value
  - 127 to 2 = linear scaling from "center" to "full reverse"
  - 1 = full "reverse"
  - 0 = disabled (i.e. **PWM** output is held low)
-

# Constructor & Destructor Documentation

## PWM::PWM ( **UINT32 channel** ) [explicit]

Allocate a **PWM** in the default module given a channel.

Using a default module allocate a **PWM** given the channel number. The default module is the first slot numerically in the cRIO chassis.

### Parameters:

**channel** The **PWM** channel on the digital module.

## PWM::PWM ( **UINT8 moduleNumber,** **UINT32 channel** )

Allocate a **PWM** given a module and channel.

Allocate a **PWM** using a module and channel number.

### Parameters:

**moduleNumber** The digital module (1 or 2).

**channel** The **PWM** channel on the digital module (1..10).

## PWM::~PWM ( ) [virtual]

Free the **PWM** channel.

Free the resource associated with the **PWM** channel and set the value to 0.

# Member Function Documentation

## **void PWM::EnableDeadbandElimination ( bool eliminateDeadband )**

Optionally eliminate the deadband from a speed controller.

### **Parameters:**

**eliminateDeadband** If true, set the motor curve on the **Jaguar** to eliminate the deadband in the middle of the range. Otherwise, keep the full range without modifying any values.

## **float PWM::GetPosition ( ) [protected, virtual]**

Get the **PWM** value in terms of a position.

This is intended to be used by servos.

### **Precondition:**

SetMaxPositivePwm() called.  
SetMinNegativePwm() called.

### **Returns:**

The position the servo is set to between 0.0 and 1.0.

## **UINT8 PWM::GetRaw ( ) [virtual]**

Get the **PWM** value directly from the hardware.

Read a raw value from a **PWM** channel.

### **Returns:**

Raw **PWM** control value. Range: 0 - 255.

## **std::string PWM::GetSmartDashboardType ( ) [protected, virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

Reimplemented in [Servo](#).

## **float PWM::GetSpeed( ) [protected, virtual]**

Get the **PWM** value in terms of speed.

This is intended to be used by speed controllers.

### **Precondition:**

SetMaxPositivePwm() called.

SetMinPositivePwm() called.

SetMaxNegativePwm() called.

SetMinNegativePwm() called.

### **Returns:**

The most recently set speed between -1.0 and 1.0.

## **ITable \* PWM::GetTable( ) [protected, virtual]**

### **Returns:**

the table that is currently associated with the sendable

Implements [Sendable](#).

Reimplemented in [Servo](#).

## **void PWM::InitTable( ITable \* subtable ) [protected, virtual]**

Initializes a table for this sendable object.

### **Parameters:**

**subtable** The table to put the values in.

Implements [Sendable](#).

Reimplemented in [Servo](#).

## **void PWM::SetBounds( INT32 max, INT32 deadbandMax, INT32 center,**

```
    INT32 deadbandMin,  
    INT32 min  
)
```

Set the bounds on the **PWM** values.

This sets the bounds on the **PWM** values for a particular each type of controller. The values determine the upper and lower speeds as well as the deadband bracket.

#### Parameters:

<b>max</b>	The Minimum pwm value
<b>deadbandMax</b>	The high end of the deadband range
<b>center</b>	The center speed (off)
<b>deadbandMin</b>	The low end of the deadband range
<b>min</b>	The minimum pwm value

### **void PWM::SetPeriodMultiplier ( PeriodMultiplier mult )**

Slow down the **PWM** signal for old devices.

#### Parameters:

<b>mult</b>	The period multiplier to apply to this channel
-------------	--

### **void PWM::SetPosition ( float pos ) [protected, virtual]**

Set the **PWM** value based on a position.

This is intended to be used by servos.

#### Precondition:

- SetMaxPositivePwm() called.
- SetMinNegativePwm() called.

#### Parameters:

<b>pos</b>	The position to set the servo between 0.0 and 1.0.
------------	--

### **void PWM::SetRaw ( UINT8 value ) [virtual]**

Set the **PWM** value directly to the hardware.

Write a raw value to a **PWM** channel.

### Parameters:

**value** Raw **PWM** value. Range 0 - 255.

**void PWM::SetSpeed ( float speed ) [protected, virtual]**

Set the **PWM** value based on a speed.

This is intended to be used by speed controllers.

### Precondition:

SetMaxPositivePwm() called.  
SetMinPositivePwm() called.  
SetCenterPwm() called.  
SetMaxNegativePwm() called.  
SetMinNegativePwm() called.

### Parameters:

**speed** The speed to set the speed controller between -1.0 and 1.0.

Reimplemented in **SafePWM**.

**void PWM::ValueChanged ( ITable \* source,  
const std::string & key,  
EntryValue value,  
bool isNew [protected, virtual]**

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

### Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

Reimplemented in **Servo**.

---

# Member Data Documentation

**const UINT32 PWM::kDefaultMinPwmHigh = 102 [static, protected]**

kDefaultMinPwmHigh is "ticks" where each tick is 6.525us

- There are 128 pwm values less than the center, so...
- The minimum output pulse length is  $1.5\text{ms} - 128 * 6.525\mu\text{s} = 0.665\text{ms}$
- $0.665\text{ms} / 6.525\mu\text{s} \text{ per tick} = 102$

**const UINT32 PWM::kDefaultPwmPeriod = 774 [static, protected]**

kDefaultPwmPeriod is "ticks" where each tick is 6.525us

- 20ms periods (50 Hz) are the "safest" setting in that this works for all devices
- 20ms periods seem to be desirable for Vex Motors
- 20ms periods are the specified period for HS-322HD servos, but work reliably down to 10.0 ms; starting at about 8.5ms, the servo sometimes hums and get hot; by 5.0ms the hum is nearly continuous
- 10ms periods work well for **Victor** 884
- 5ms periods allows higher update rates for Luminary Micro **Jaguar** speed controllers. Due to the shipping firmware on the **Jaguar**, we can't run the update period less than 5.05 ms.

kDefaultPwmPeriod is the 1x period (5.05 ms). In hardware, the period scaling is implemented as an output squelch to get longer periods for old devices.

Set to 5.05 ms period / 6.525us clock = 774

---

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

- C:/WindRiver/workspace/WPIlib/**PWM.h**
- C:/WindRiver/workspace/WPIlib/PWM.cpp



# ReentrantSemaphore Class Reference

---

Wrap a vxWorks semaphore (SEM\_ID) for easier use in C++. More...

```
#include <Synchronized.h>
```

List of all members.

## Public Member Functions

int **take ()**

Lock the semaphore, blocking until it's available.

int **give ()**

Unlock the semaphore.

class **Synchronized**

---

## Detailed Description

Wrap a vxWorks semaphore (SEM\_ID) for easier use in C++.

For a static instance, the constructor runs at program load time before main() can spawn any tasks. Use that to fix race conditions in setup code.

This uses a semM semaphore which is "reentrant" in the sense that the owning task can "take" the semaphore more than once. It will need to "give" the semaphore the same number of times to unlock it.

This class is safe to use in static variables because it does not depend on any other C++ static constructors or destructors.

---

# Member Function Documentation

## **int ReentrantSemaphore::give( )** [inline]

Unlock the semaphore.

### **Returns:**

0 for success, -1 for error. If -1, the error will be in errno.

## **int ReentrantSemaphore::take( )** [inline]

Lock the semaphore, blocking until it's available.

### **Returns:**

0 for success, -1 for error. If -1, the error will be in errno.

---

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

- C:/WindRiver/workspace/WPILib/**Synchronized.h**

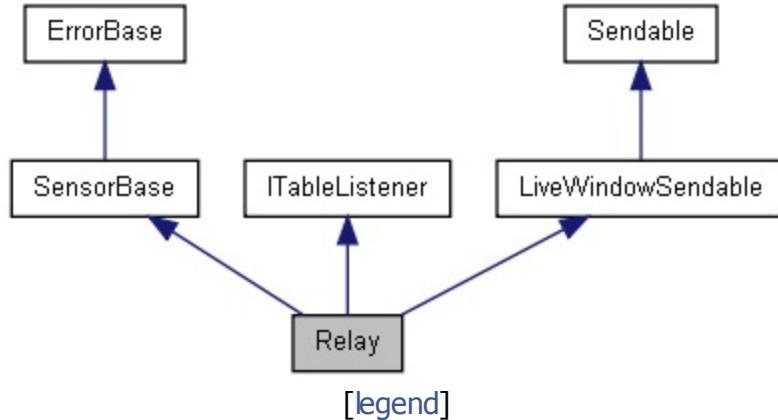


# Relay Class Reference

Class for Spike style relay outputs. [More...](#)

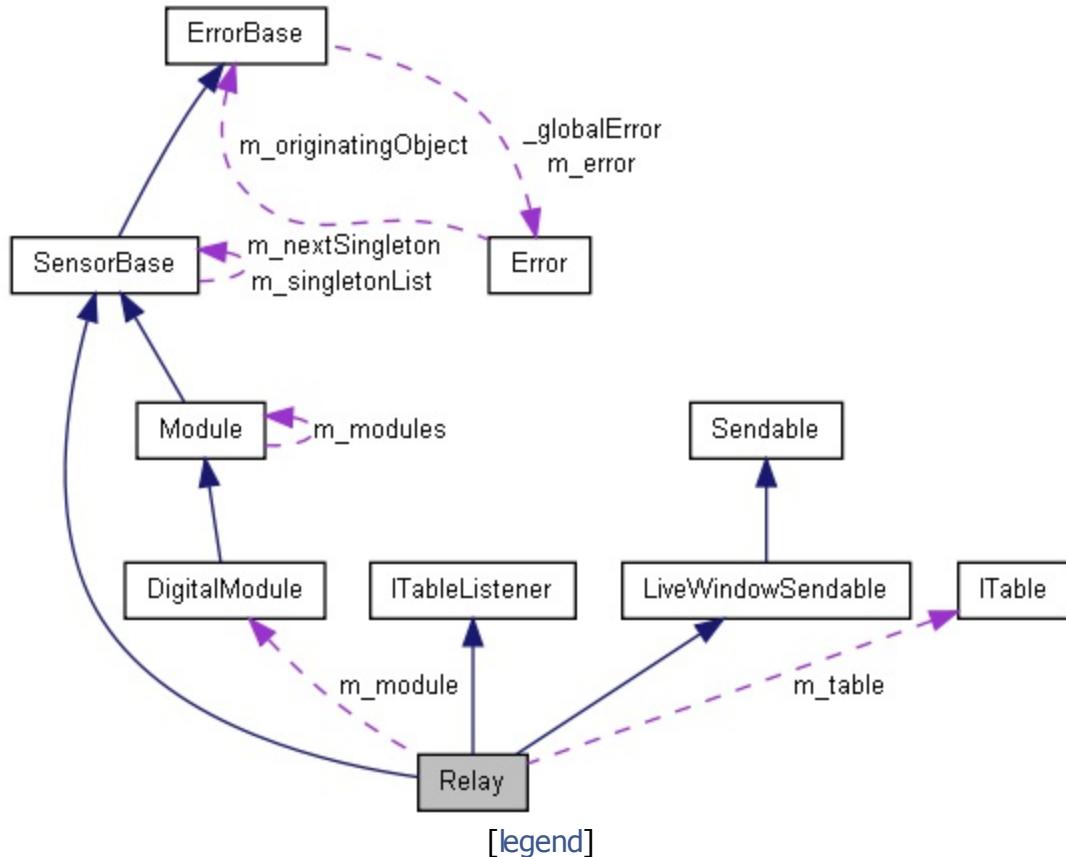
```
#include <Relay.h>
```

Inheritance diagram for Relay:



[legend]

Collaboration diagram for Relay:



[legend]

List of all members.

# Public Types

enum **Value** { **kOff**, **kOn**, **kForward**, **kReverse** }

enum **Direction** { **kBothDirections**, **kForwardOnly**, **kReverseOnly** }

**Relay** (UINT32 channel, Direction direction=kBothDirections)

**Relay** constructor given a channel only where the default digital module is used.

**Relay** (UINT8 moduleNumber, UINT32 channel, Direction direction=kBothDirections)

**Relay** constructor given the module and the channel.

virtual **~Relay** ()

Free the resource associated with a relay.

void **Set** (Value value)

Set the relay state.

Value **Get** ()

Get the **Relay** State.

void **ValueChanged** (ITable \*source, const std::string &key, EntryValue value, bool isNew)

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

void **UpdateTable** ()

Update the table for this sendable object with the latest values.

void **StartLiveWindowMode** ()

Start having this sendable object automatically respond to value changes reflect the value on the table.

void **StopLiveWindowMode** ()

Stop having this sendable object automatically respond to value changes.

std::string **GetSmartDashboardType** ()

void **InitTable** (ITable \*subTable)

Initializes a table for this sendable object.

**ITable** \* **GetTable** ()

## Public Attributes

**ITable \* m\_table**

---

## Detailed Description

Class for Spike style relay outputs.

Relays are intended to be connected to spikes or similar relays. The relay channels controls a pair of pins that are either both off, one on, the other on, or both on. This translates into two spike outputs at 0v, one at 12v and one at 0v, one at 0v and the other at 12v, or two spike outputs at 12V. This allows off, full forward, or full reverse control of motors without variable speed. It also allows the two channels (forward and reverse) to be used independently for something that does not care about voltage polarity (like a solenoid).

---

# Constructor & Destructor Documentation

```
Relay::Relay ( UINT32      channel,  
              Relay::Direction direction = kBothDirections  
            )
```

**Relay** constructor given a channel only where the default digital module is used.

## Parameters:

**channel** The channel number within the default module for this relay.  
**direction** The direction that the **Relay** object will control.

```
Relay::Relay ( UINT8      moduleNumber,  
              UINT32      channel,  
              Relay::Direction direction = kBothDirections  
            )
```

**Relay** constructor given the module and the channel.

## Parameters:

**moduleNumber** The digital module this relay is connected to (1 or 2).  
**channel** The channel number within the module for this relay.  
**direction** The direction that the **Relay** object will control.

```
Relay::~Relay ( ) [virtual]
```

Free the resource associated with a relay.

The relay channels are set to free and the relay output is turned off.

# Member Function Documentation

## **R**elay::Value **R**elay::Get( )

Get the **R**elay State.

Gets the current state of the relay.

When set to kForwardOnly or kReverseOnly, value is returned as kOn/kOff not kForward/kReverse (per the recommendation in Set)

### Returns:

The current state of the relay as a Relay::Value

## **I**std::string **R**elay::GetSmartDashboardType( ) [virtual]

### Returns:

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

## **I**Table \* **R**elay::GetTable( ) [virtual]

### Returns:

the table that is currently associated with the sendable

Implements **Sendable**.

## **void** **R**elay::InitTable( **I**Table \* **subtable** ) [virtual]

Initializes a table for this sendable object.

### Parameters:

**subtable** The table to put the values in.

Implements **Sendable**.

## **void** **R**elay::Set( **R**elay::Value **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 be any of the four states: 0v-0v, 0v-12v, 12v-0v, 12v-12v

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:

**value** The state to set the relay.

```
void Relay::ValueChanged( ITable * source,  
                           const std::string & key,  
                           EntryValue value,  
                           bool isNew  
                         )  
                           [virtual]
```

Called when a key-value pair is changed in a **ITable** **WARNING:** If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

### Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

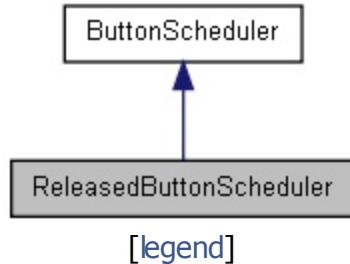
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/**Relay.h**
- C:/WindRiver/workspace/WPILib/Relay.cpp

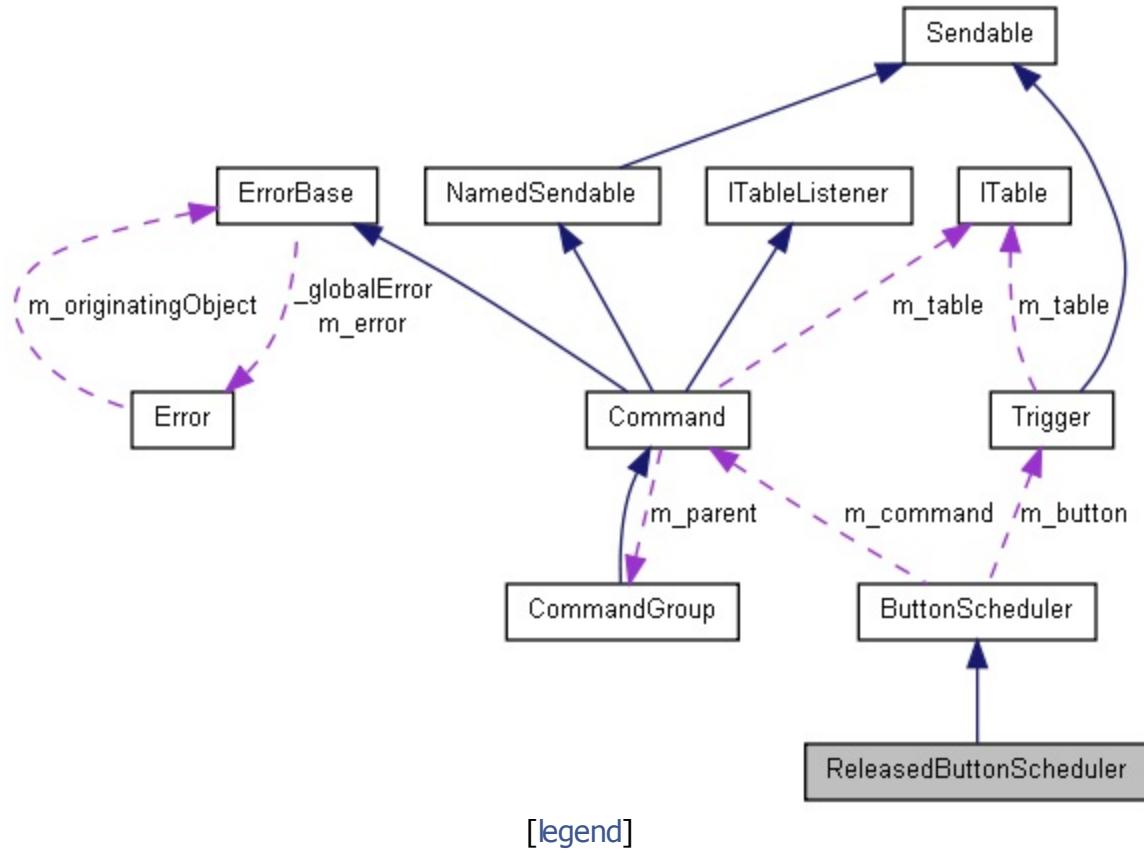


# ReleasedButtonScheduler Class Reference

Inheritance diagram for ReleasedButtonScheduler:



Collaboration diagram for ReleasedButtonScheduler:



List of all members.

# Public Member Functions

**ReleasedButtonScheduler** (bool last, **Trigger** \*button, **Command** \*orders)

virtual void **Execute** ()

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

- C:/WindRiver/workspace/WPILib/Buttons/**ReleasedButtonScheduler.h**
- C:/WindRiver/workspace/WPILib/Buttons/ReleasedButtonScheduler.cpp

Generated by  1.7.2

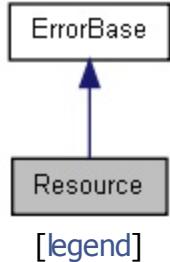
[Public Member Functions](#) |  
[Static Public Member Functions](#)

# Resource Class Reference

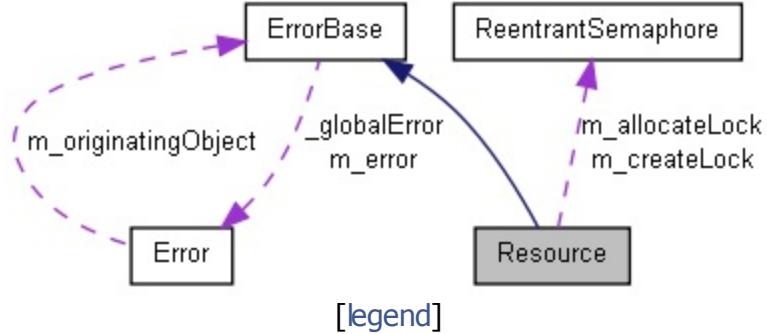
The **Resource** class is a convenient way to track allocated resources. [More...](#)

```
#include <Resource.h>
```

Inheritance diagram for Resource:



Collaboration diagram for Resource:



List of all members.

# Public Member Functions

virtual	<b>~Resource ()</b>
Delete the allocated array or resources.	
UINT32	<b>Allocate (const char *resourceDesc)</b>
Allocate a resource.	
UINT32	<b>Allocate (UINT32 index, const char *resourceDesc)</b>
Allocate a specific resource value.	
void	<b>Free (UINT32 index)</b>
Free an allocated resource.	

---

static void **CreateResourceObject (Resource \*\*r, UINT32 elements)**  
Factory method to create a **Resource** allocation-tracker \*if\* needed.

## Detailed Description

The **Resource** class is a convenient way to track allocated resources.

It tracks them as indices in the range [0 .. elements - 1]. E.g. the library uses this to track hardware channel allocation.

The **Resource** class does not allocate the hardware channels or other resources; it just tracks which indices were marked in use by Allocate and not yet freed by Free.

---

# Constructor & Destructor Documentation

## **Resource::~Resource( ) [virtual]**

Delete the allocated array or resources.

This happens when the module is unloaded (provided it was statically allocated).

# Member Function Documentation

## **UINT32 Resource::Allocate ( const char \* resourceDesc )**

Allocate a resource.

When a resource is requested, mark it allocated. In this case, a free resource value within the range is located and returned after it is marked allocated.

## **UINT32 Resource::Allocate ( **UINT32 index,**                           **const char \* resourceDesc** )**

Allocate a specific resource value.

The user requests a specific resource value, i.e. channel number and it is verified unallocated, then returned.

## **void Resource::CreateResourceObject ( **Resource \*\* r,**                           **UINT32 elements** )                    [static]**

Factory method to create a **Resource** allocation-tracker \*if\* needed.

### **Parameters:**

- r** -- address of the caller's **Resource** pointer. If **\*r == NULL**, this will construct a **Resource** and make **\*r** point to it. If **\*r != NULL**, i.e. the caller already has a **Resource** instance, this won't do anything.
- elements** -- the number of elements for this **Resource** allocator to track, that is, it will allocate resource numbers in the range [0 .. elements - 1].

## **void Resource::Free ( **UINT32 index** )**

Free an allocated resource.

After a resource is no longer needed, for example a destructor is called for a channel assignment class, Free will release the resource value so it can be reused somewhere else in the program.

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

- C:/WindRiver/workspace/WPILib/**Resource.h**
- C:/WindRiver/workspace/WPILib/Resource.cpp

---

Generated by  1.7.2



# HiTechnicColorSensor::RGB Struct Reference

---

List of all members.

## Public Attributes

UINT16	<b>red</b>
UINT16	<b>blue</b>
UINT16	<b>green</b>

---

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

- C:/WindRiver/workspace/WPILib/**HiTechnicColorSensor.h**

Generated by  1.7.2

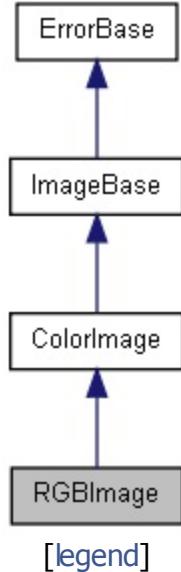


# RGBImage Class Reference

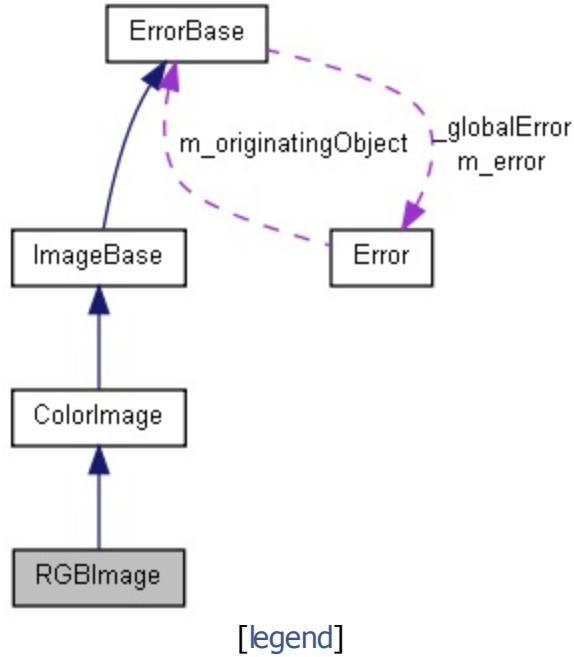
A color image represented in RGB color space at 3 bytes per pixel. [More...](#)

```
#include <RGBImage.h>
```

Inheritance diagram for RGBImage:



Collaboration diagram for RGBImage:



[List of all members.](#)

## Public Member Functions

### **RGBImage ()**

Create a new image that uses Red, Green, and Blue planes.

### **RGBImage (const char \*fileName)**

Create a new image by loading a file.

---

## Detailed Description

A color image represented in RGB color space at 3 bytes per pixel.

---

# Constructor & Destructor Documentation

## **RGBImage::RGBImage ( const char \* fileName )**

Create a new image by loading a file.

### Parameters:

**fileName** The path of the file to load.

---

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

- C:/WindRiver/workspace/WPILib/Vision/**RGBImage.h**
- C:/WindRiver/workspace/WPILib/Vision/RGBImage.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

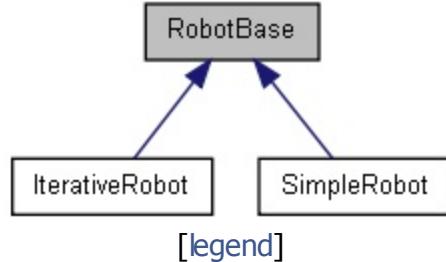
[Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#) | [Friends](#)

# RobotBase Class Reference

Implement a Robot Program framework. More...

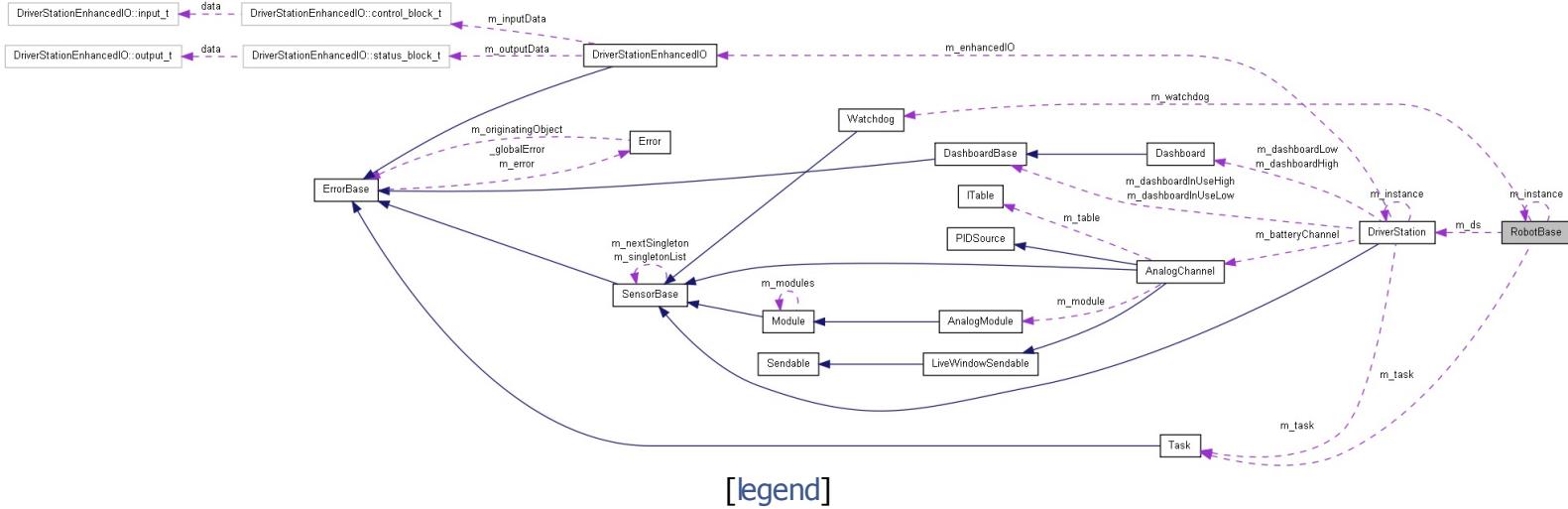
```
#include <RobotBase.h>
```

Inheritance diagram for RobotBase:



[legend]

Collaboration diagram for RobotBase:



[legend]

List of all members.

bool **IsEnabled ()**

Determine if the Robot is currently enabled.

bool **IsDisabled ()**

Determine if the Robot is currently disabled.

bool **IsAutonomous ()**

Determine if the robot is currently in Autonomous mode.

bool **IsOperatorControl ()**

Determine if the robot is currently in Operator Control mode.

bool **IsTest ()**

Determine if the robot is currently in Test mode.

bool **IsSystemActive ()**

Check on the overall status of the system.

bool **IsNewDataAvailable ()**

Indicates if new data is available from the driver station.

**Watchdog** & **GetWatchdog ()**

Return the instance of the **Watchdog** timer.

static **RobotBase** & **getInstance ()**

static void **setInstance (RobotBase \*robot)**

static void **startRobotTask (FUNCPTR factory)**

Start the robot code.

static void **robotTask (FUNCPTR factory, Task \*task)**

Static interface that will start the competition in the new task.

# Protected Member Functions

virtual ~**RobotBase** ()

Free the resources for a **RobotBase** class.

virtual void **StartCompetition** ()=0

**RobotBase** ()

Constructor for a generic robot program.

**Task** \* **m\_task**

**Watchdog** **m\_watchdog**

**DriverStation** \* **m\_ds**

class **RobotDeleter**

## Detailed Description

Implement a Robot Program framework.

The **RobotBase** class is intended to be subclassed by a user creating a robot program. Overridden Autonomous() and OperatorControl() methods are called at the appropriate time as the match proceeds. In the current implementation, the Autonomous code will run to completion before the OperatorControl code could start. In the future the Autonomous code might be spawned as a task, then killed at the end of the Autonomous period.

---

# Constructor & Destructor Documentation

## **RobotBase::~RobotBase( )** [protected, virtual]

Free the resources for a **RobotBase** class.

This includes deleting all classes that might have been allocated as Singletons to they would never be deleted except here.

## **RobotBase::RobotBase( )** [protected]

Constructor for a generic robot program.

User code should be placed in the constructor that runs before the Autonomous or Operator Control period starts. The constructor will run to completion before Autonomous is entered.

This must be used to ensure that the communications code starts. In the future it would be nice to put this code into it's own task that loads on boot so ensure that it runs.

# Member Function Documentation

## **Watchdog & RobotBase::GetWatchdog( )**

Return the instance of the **Watchdog** timer.

Get the watchdog timer so the user program can either disable it or feed it when necessary.

## **bool RobotBase::IsAutonomous( )**

Determine if the robot is currently in Autnomous mode.

### **Returns:**

True if the robot is currently operating Autonomously as determined by the field controls.

## **bool RobotBase::IsDisabled( )**

Determine if the Robot is currently disabled.

### **Returns:**

True if the Robot is currently disabled by the field controls.

## **bool RobotBase::IsEnabled( )**

Determine if the Robot is currently enabled.

### **Returns:**

True if the Robot is currently enabled by the field controls.

## **bool RobotBase::IsNewDataAvailable( )**

Indicates if new data is available from the driver station.

### **Returns:**

Has new data arrived over the network since the last time this function was called?

## **bool RobotBase::IsOperatorControl( )**

Determine if the robot is currently in Operator Control mode.

### **Returns:**

True if the robot is currently operating in Tele-Op mode as determined by the field controls.

## **bool RobotBase::IsSystemActive( )**

Check on the overall status of the system.

### **Returns:**

Is the system active (i.e. **PWM** motor outputs, etc. enabled)?

## **bool RobotBase::IsTest( )**

Determine if the robot is currently in Test mode.

### **Returns:**

True if the robot is currently running tests as determined by the field controls.

## **void RobotBase::startRobotTask( FUNCPTR factory ) [static]**

Start the robot code.

This function starts the robot code running by spawning a task. Currently tasks seemed to be started by LVRT without setting the VX\_FP\_TASK flag so floating point context is not saved on interrupts. Therefore the program experiences hard to debug and unpredictable results. So the LVRT code starts this function, and it, in turn, starts the actual user program.

---

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

- C:/WindRiver/workspace/WPIlib/**RobotBase.h**
- C:/WindRiver/workspace/WPIlib/RobotBase.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

# RobotDelete Class Reference

---

This class exists for the sole purpose of getting its destructor called when the module unloads. [More...](#)

[List of all members.](#)

---

## Detailed Description

This class exists for the sole purpose of getting its destructor called when the module unloads.

Before the module is done unloading, we need to delete the **RobotBase** derived singleton. This should delete the other remaining singletons that were registered. This should also stop all tasks that are using the **Task** class.

---

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

- C:/WindRiver/workspace/WPIlib/RobotBase.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

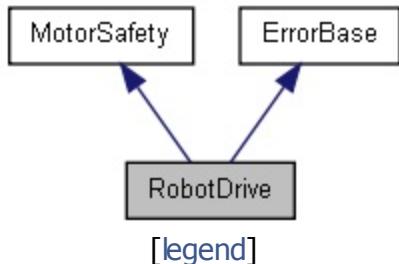
[Public Types](#) | [Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#) |  
[Static Protected Attributes](#)

# RobotDrive Class Reference

Utility class for handling Robot drive based on a definition of the motor configuration.  
[More...](#)

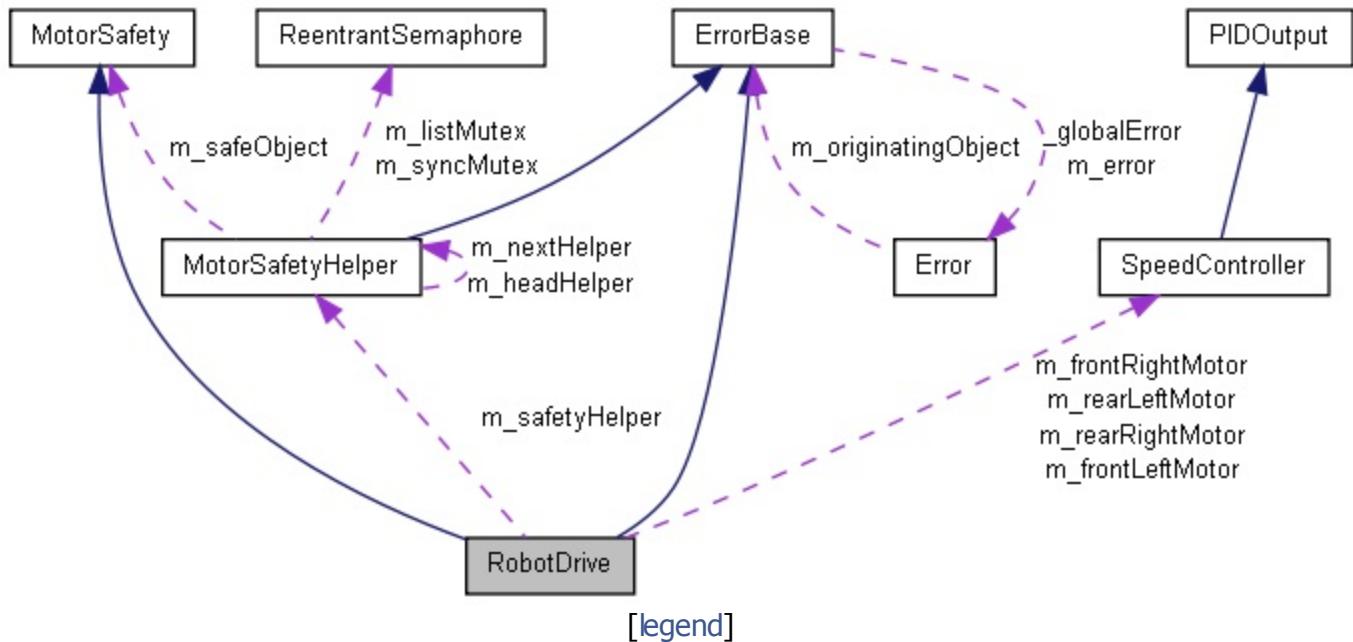
```
#include <RobotDrive.h>
```

Inheritance diagram for RobotDrive:



[legend]

Collaboration diagram for RobotDrive:



[legend]

List of all members.

# Public Types

void	<b>TankDrive</b> ( <b>GenericHID</b> *leftStick, <b>GenericHID</b> *rightStick, bool squaredInputs=true)	Provide tank steering using the stored robot configuration.
void	<b>TankDrive</b> ( <b>GenericHID</b> &leftStick, <b>GenericHID</b> &rightStick, bool squaredInputs=true)	
void	<b>TankDrive</b> ( <b>GenericHID</b> *leftStick, <b>UINT32</b> leftAxis, <b>GenericHID</b> *rightStick, <b>UINT32</b> rightAxis, bool squaredInputs=true)	Provide tank steering using the stored robot configuration.
void	<b>TankDrive</b> ( <b>GenericHID</b> &leftStick, <b>UINT32</b> leftAxis, <b>GenericHID</b> &rightStick, <b>UINT32</b> rightAxis, bool squaredInputs=true)	
void	<b>TankDrive</b> (float leftValue, float rightValue, bool squaredInputs=true)	Provide tank steering using the stored robot configuration.
void	<b>ArcadeDrive</b> ( <b>GenericHID</b> *stick, bool squaredInputs=true)	Arcade drive implements single stick driving.
void	<b>ArcadeDrive</b> ( <b>GenericHID</b> &stick, bool squaredInputs=true)	Arcade drive implements single stick driving.
void	<b>ArcadeDrive</b> ( <b>GenericHID</b> *moveStick, <b>UINT32</b> moveChannel, <b>GenericHID</b> *rotateStick, <b>UINT32</b> rotateChannel, bool squaredInputs=true)	Arcade drive implements single stick driving.
void	<b>ArcadeDrive</b> ( <b>GenericHID</b> &moveStick, <b>UINT32</b> moveChannel, <b>GenericHID</b> &rotateStick, <b>UINT32</b> rotateChannel, bool squaredInputs=true)	Arcade drive implements single stick driving.
void	<b>ArcadeDrive</b> (float moveValue, float rotateValue, bool squaredInputs=true)	Arcade drive implements single stick driving.
void	<b>MecanumDrive_Cartesian</b> (float x, float y, float rotation, float gyroAngle=0.0)	Drive method for Mecanum wheeled robots.
void	<b>MecanumDrive_Polar</b> (float magnitude, float direction, float rotation)	Drive method for Mecanum wheeled robots.
void	<b>HolonomicDrive</b> (float magnitude, float direction, float rotation)	Holonomic Drive method for Mecanum wheeled robots.

virtual void	<b>SetLeftRightMotorOutputs</b> (float leftOutput, float rightOutput) Set the speed of the right and left motors.
void	<b>SetInvertedMotor</b> (MotorType motor, bool isInverted)
void	<b>SetSensitivity</b> (float sensitivity) Set the turning sensitivity.
void	<b>SetMaxOutput</b> (double maxOutput) Configure the scaling factor for using <b>RobotDrive</b> with motor controllers in a mode other than PercentVbus.
void	<b>SetExpiration</b> (float timeout)
float	<b>GetExpiration</b> ()
bool	<b>IsAlive</b> ()
void	<b>StopMotor</b> ()
bool	<b>IsSafetyEnabled</b> ()
void	<b>SetSafetyEnabled</b> (bool enabled)
void	<b>GetDescription</b> (char *desc)

void	<b>InitRobotDrive</b> () Common function to initialize all the robot drive constructors.
float	<b>Limit</b> (float num) Limit motor values to the -1.0 to +1.0 range.
void	<b>Normalize</b> (double *wheelSpeeds) Normalize all wheel speeds if the magnitude of any wheel is greater than 1.0.
void	<b>RotateVector</b> (double &x, double &y, double angle) Rotate a vector in Cartesian space.

## Protected Attributes

```
INT32 m_invertedMotors [kMaxNumberOfMotors]
    float m_sensitivity
    double m_maxOutput
    bool m_deleteSpeedControllers
    SpeedController * m_frontLeftMotor
    SpeedController * m_frontRightMotor
    SpeedController * m_rearLeftMotor
    SpeedController * m_rearRightMotor
    MotorSafetyHelper * m_safetyHelper
```

## Static Protected Attributes

```
static const INT32 kMaxNumberOfMotors = 4
```

## Detailed Description

Utility class for handling Robot drive based on a definition of the motor configuration.

The robot drive class handles basic driving for a robot. Currently, 2 and 4 motor standard drive trains are supported. In the future other drive types like swerve and meccanum might be implemented. Motor channel numbers are passed supplied on creation of the class. Those are used for either the Drive function (intended for hand created drive code, such as autonomous) or with the Tank/Arcade functions intended to be used for Operator Control driving.

---

# Constructor & Destructor Documentation

```
RobotDrive::RobotDrive ( UINT32 leftMotorChannel,  
                        UINT32 rightMotorChannel  
)
```

Constructor for **RobotDrive** with 2 motors specified with channel numbers.

Set up parameters for a two wheel drive system where the left and right motor pwm channels are specified in the call. This call assumes Jaguars for controlling the motors.

## Parameters:

- leftMotorChannel** The **PWM** channel number on the default digital module that drives the left motor.
- rightMotorChannel** The **PWM** channel number on the default digital module that drives the right motor.

```
RobotDrive::RobotDrive ( UINT32 frontLeftMotor,  
                        UINT32 rearLeftMotor,  
                        UINT32 frontRightMotor,  
                        UINT32 rearRightMotor  
)
```

Constructor for **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

```
RobotDrive::RobotDrive ( SpeedController * leftMotor,
```

```
SpeedController * rightMotor
```

```
)
```

Constructor for **RobotDrive** with 2 motors specified as **SpeedController** objects.

The **SpeedController** version of the constructor enables programs to use the **RobotDrive** classes with subclasses of the **SpeedController** objects, for example, versions with ramping or reshaping of the curve to suit motor bias or deadband elimination.

#### Parameters:

**leftMotor** The left **SpeedController** object used to drive the robot.

**rightMotor** the right **SpeedController** object used to drive the robot.

```
RobotDrive::RobotDrive ( SpeedController * frontLeftMotor,
```

```
        SpeedController * rearLeftMotor,
```

```
        SpeedController * frontRightMotor,
```

```
        SpeedController * rearRightMotor
```

```
)
```

Constructor for **RobotDrive** with 4 motors specified as **SpeedController** objects.

Speed controller input version of **RobotDrive** (see previous comments).

#### Parameters:

**rearLeftMotor** The back left **SpeedController** object used to drive the robot.

**frontLeftMotor** The front left **SpeedController** object used to drive the robot

**rearRightMotor** The back right **SpeedController** object used to drive the robot.

**frontRightMotor** The front right **SpeedController** object used to drive the robot.

```
RobotDrive::~RobotDrive ( ) [virtual]
```

**RobotDrive** destructor.

Deletes motor objects that were not passed in and created internally only.



# Member Function Documentation

```
void RobotDrive::ArcadeDrive( GenericHID * stick,  
                                bool          squaredInputs = true  
                            )
```

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:

**stick** 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

```
void RobotDrive::ArcadeDrive( GenericHID & stick,  
                                bool          squaredInputs = true  
                            )
```

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:

**stick** 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

```
void RobotDrive::ArcadeDrive( GenericHID * moveStick,  
                               UINT32        moveAxis,  
                               GenericHID * rotateStick,  
                               UINT32        rotateAxis,  
                               bool          squaredInputs = true  
                           )
```

)

Arcade drive implements single stick driving.

Given two joystick instances and two axis, compute the values to send to either two or four motors.

#### Parameters:

<b>moveStick</b>	The <b>Joystick</b> object that represents the forward/backward direction
<b>moveAxis</b>	The axis on the moveStick object to use for fowards/backwards (typically Y_AXIS)
<b>rotateStick</b>	The <b>Joystick</b> object that represents the rotation value
<b>rotateAxis</b>	The axis on the rotation object to use for the rotate right/left (typically X_AXIS)
<b>squaredInputs</b>	Setting this parameter to true increases the sensitivity at lower speeds

```
void RobotDrive::ArcadeDrive( GenericHID & moveStick,  
                                UINT32      moveAxis,  
                                GenericHID & rotateStick,  
                                UINT32      rotateAxis,  
                                bool        squaredInputs = true  
                            )
```

Arcade drive implements single stick driving.

Given two joystick instances and two axis, compute the values to send to either two or four motors.

#### Parameters:

<b>moveStick</b>	The <b>Joystick</b> object that represents the forward/backward direction
<b>moveAxis</b>	The axis on the moveStick object to use for fowards/backwards (typically Y_AXIS)
<b>rotateStick</b>	The <b>Joystick</b> object that represents the rotation value
<b>rotateAxis</b>	The axis on the rotation object to use for the rotate right/left (typically X_AXIS)
<b>squaredInputs</b>	Setting this parameter to true increases the sensitivity at lower speeds

```
void RobotDrive::ArcadeDrive ( float moveValue,
                                float rotateValue,
                                bool squaredInputs = true
                            )
```

Arcade drive implements single stick driving.

This function lets you directly provide joystick values from any source.

#### Parameters:

**moveValue** The value to use for forwards/backwards  
**rotateValue** The value to use for the rotate right/left  
**squaredInputs** If set, increases the sensitivity at low speeds

```
void RobotDrive::Drive ( float outputMagnitude,
                        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 will most likely be used in an autonomous routine.

#### Parameters:

**outputMagnitude** The forward component of the output magnitude to send to the motors.  
**curve** The rate of turn, constant for different forward speeds.

```
void RobotDrive::HolonomicDrive ( float magnitude,
                                    float direction,
                                    float rotation
                                )
```

Holonomic Drive method for Mecanum wheeled robots.

This is an alias to [\*\*MecanumDrive\\_Polar\(\)\*\*](#) for backward compatibility

## Parameters:

- magnitude** The speed that the robot should drive in a given direction. [-1.0..1.0]
- direction** The direction the robot should drive. The direction and magnitude are independent of the rotation rate.
- rotation** The rate of rotation for the robot that is completely independent of the magnitude or direction. [-1.0..1.0]

## **void RobotDrive::InitRobotDrive( ) [protected]**

Common function to initialize all the robot drive constructors.

Create a motor safety object (the real reason for the common code) and initialize all the motor assignments. The default timeout is set for the robot drive.

## **void RobotDrive::MecanumDrive\_Cartesian( float x, float y, float rotation, float gyroAngle = 0.0 )**

Drive method for Mecanum wheeled robots.

A method for driving with Mecanum wheeled robots. There are 4 wheels on the robot, arranged so that the front and back wheels are toed in 45 degrees. When looking at the wheels from the top, the roller axles should form an X across the robot.

This is designed to be directly driven by joystick axes.

## Parameters:

- x** The speed that the robot should drive in the X direction. [-1.0..1.0]
- y** The speed that the robot should drive in the Y direction. This input is inverted to match the forward == -1.0 that joysticks produce. [-1.0..1.0]
- rotation** The rate of rotation for the robot that is completely independent of the translation. [-1.0..1.0]
- gyroAngle** The current angle reading from the gyro. Use this to implement field-oriented controls.

```
void RobotDrive::MecanumDrive_Polar ( float magnitude,  
                                     float direction,  
                                     float rotation  
                                     )
```

Drive method for Mecanum wheeled robots.

A method for driving with Mecanum wheeled robots. There are 4 wheels on the robot, arranged so that the front and back wheels are toed in 45 degrees. When looking at the wheels from the top, the roller axles should form an X across the robot.

#### Parameters:

- magnitude** The speed that the robot should drive in a given direction. [-1.0..1.0]
- direction** The direction the robot should drive in degrees. The direction and magnitude are independent of the rotation rate.
- rotation** The rate of rotation for the robot that is completely independent of the magnitude or direction. [-1.0..1.0]

```
void RobotDrive::SetLeftRightMotorOutputs ( float leftOutput,  
                                            float rightOutput  
                                            ) [virtual]
```

Set the speed of the right and left motors.

This is used once an appropriate drive setup function is called such as TwoWheelDrive(). The motors are set to "leftOutput" and "rightOutput" and includes flipping the direction of one side for opposing motors.

#### Parameters:

- leftOutput** The speed to send to the left side of the robot.
- rightOutput** The speed to send to the right side of the robot.

```
void RobotDrive::SetMaxOutput ( double maxOutput )
```

Configure the scaling factor for using **RobotDrive** with motor controllers in a mode other than PercentVbus.

## Parameters:

**maxOutput** Multiplied with the output percentage computed by the drive functions.

## **void RobotDrive::SetSensitivity ( float sensitivity )**

Set the turning sensitivity.

This only impacts the **Drive()** entry-point.

## Parameters:

**sensitivity** Effectively sets the turning sensitivity (or turn radius for a given value)

## **void RobotDrive::TankDrive ( float leftValue, float rightValue, bool squaredInputs = true )**

Provide tank steering using the stored robot configuration.

This function lets you directly provide joystick values from any source.

## Parameters:

**leftValue** The value of the left stick.

**rightValue** The value of the right stick.

## **void RobotDrive::TankDrive ( GenericHID \* leftStick, GenericHID \* rightStick, bool squaredInputs = true )**

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:

**leftStick** The joystick to control the left side of the robot.

**rightStick** The joystick to control the right side of the robot.

```
void RobotDrive::TankDrive( GenericHID * leftStick,
                            UINT32          leftAxis,
                            GenericHID * rightStick,
                            UINT32          rightAxis,
                            bool            squaredInputs = true
)
```

Provide tank steering using the stored robot configuration.

This function lets you pick the axis to be used on each **Joystick** object for the left and right sides of the robot.

#### Parameters:

- leftStick** The **Joystick** object to use for the left side of the robot.
- leftAxis** The axis to select on the left side **Joystick** object.
- rightStick** The **Joystick** object to use for the right side of the robot.
- rightAxis** The axis to select on the right side **Joystick** object.

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

- C:/WindRiver/workspace/WPILib/**RobotDrive.h**
- C:/WindRiver/workspace/WPILib/RobotDrive.cpp

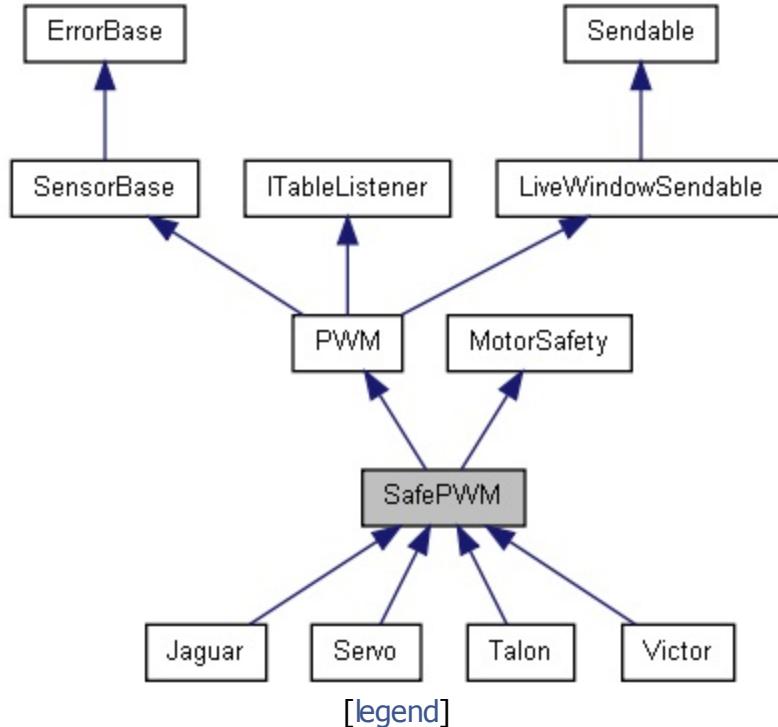


# SafePWM Class Reference

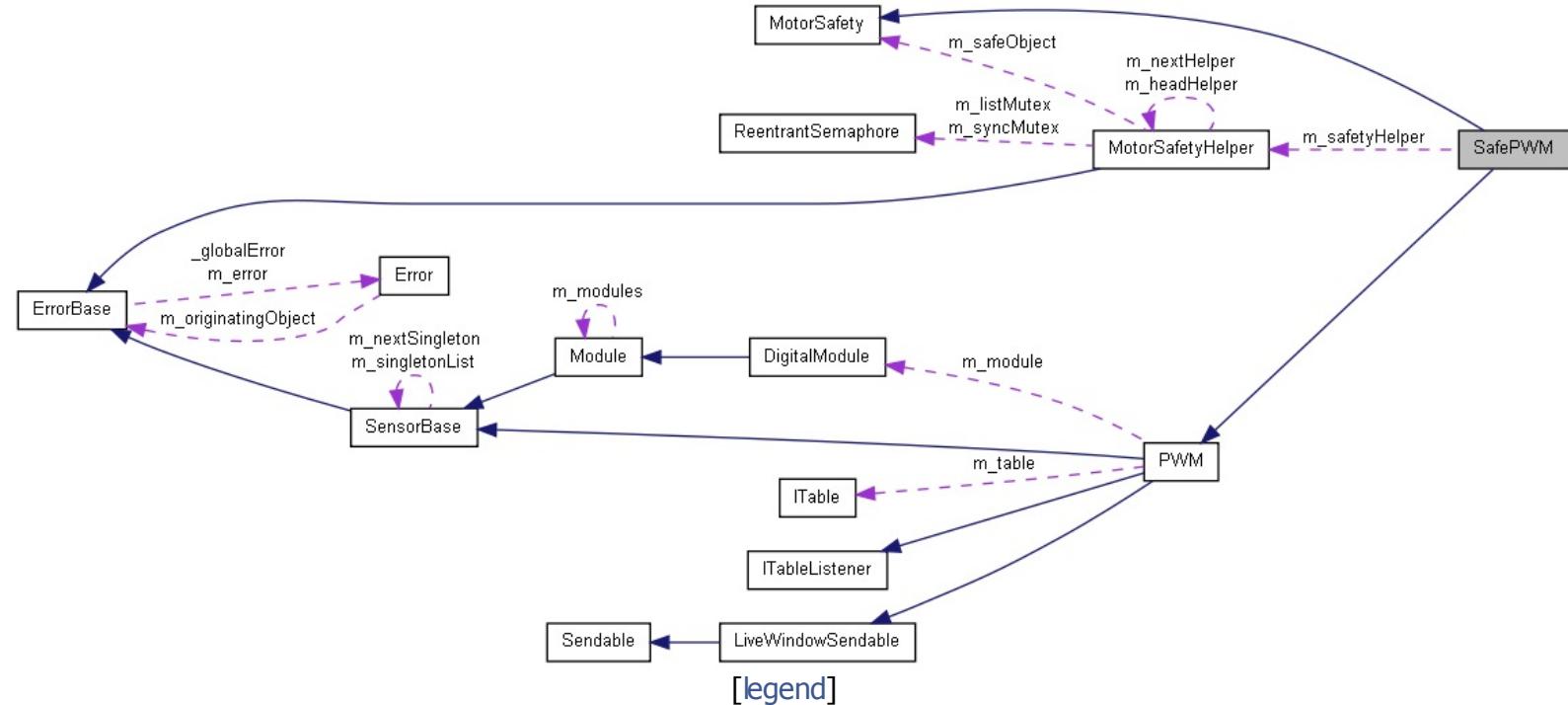
A safe version of the **PWM** class. More...

```
#include <SafePWM.h>
```

Inheritance diagram for SafePWM:



Collaboration diagram for SafePWM:



List of all members.

# Public Member Functions

**SafePWM** (UINT32 channel)

Constructor for a **SafePWM** object taking a channel number.

**SafePWM** (UINT8 moduleNumber, UINT32 channel)

Constructor for a **SafePWM** object taking channel and slot numbers.

void **SetExpiration** (float timeout)

float **GetExpiration** ()

Return the expiration time for the **PWM** object.

bool **IsAlive** ()

Check if the **PWM** object is currently alive or stopped due to a timeout.

void **StopMotor** ()

Stop the motor associated with this **PWM** object.

bool **IsSafetyEnabled** ()

Check if motor safety is enabled for this object.

void **SetSafetyEnabled** (bool enabled)

Enable/disable motor safety for this device Turn on and off the motor safety option for this **PWM** object.

void **GetDescription** (char \*desc)

virtual void **SetSpeed** (float speed)

Feed the **MotorSafety** timer when setting the speed.

## Detailed Description

A safe version of the [PWM](#) class.

It is safe because it implements the [MotorSafety](#) interface that provides timeouts in the event that the motor value is not updated before the expiration time. This delegates the actual work to a [MotorSafetyHelper](#) object that is used for all objects that implement [MotorSafety](#).

---

# Constructor & Destructor Documentation

## **SafePWM::SafePWM ( **UINT32 channel** ) [explicit]**

Constructor for a **SafePWM** object taking a channel number.

### Parameters:

**channel** The channel number to be used for the underlying **PWM** object

## **SafePWM::SafePWM ( **UINT8 moduleNumber,**                  **UINT32 channel** )**

Constructor for a **SafePWM** object taking channel and slot numbers.

### Parameters:

**moduleNumber** The digital module (1 or 2).

**channel** The **PWM** channel number on the module (1..10).

# Member Function Documentation

## **float SafePWM::GetExpiration( ) [virtual]**

Return the expiration time for the **PWM** object.

### **Returns:**

The expiration time value.

Implements **MotorSafety**.

## **bool SafePWM::IsAlive( ) [virtual]**

Check if the **PWM** object is currently alive or stopped due to a timeout.

### **Returns:**

a bool value that is true if the motor has NOT timed out and should still be running.

Implements **MotorSafety**.

## **bool SafePWM::IsSafetyEnabled( ) [virtual]**

Check if motor safety is enabled for this object.

### **Returns:**

True if motor safety is enforced for this object

Implements **MotorSafety**.

## **void SafePWM::SetSafetyEnabled( bool enabled ) [virtual]**

Enable/disable motor safety for this device Turn on and off the motor safety option for this **PWM** object.

### **Parameters:**

**enabled** True if motor safety is enforced for this object

Implements **MotorSafety**.

## **void SafePWM::SetSpeed ( float speed ) [virtual]**

Feed the **MotorSafety** timer when setting the speed.

This method is called by the subclass motor whenever it updates its speed, thereby resetting the timeout value.

### **Parameters:**

**speed** Value to pass to the **PWM** class

Reimplemented from **PWM**.

## **void SafePWM::StopMotor ( ) [virtual]**

Stop the motor associated with this **PWM** object.

This is called by the **MotorSafetyHelper** object when it has a timeout for this **PWM** and needs to stop it from running.

Implements **MotorSafety**.

---

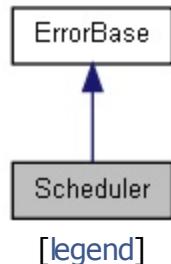
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPIlib/**SafePWM.h**
- C:/WindRiver/workspace/WPIlib/SafePWM.cpp

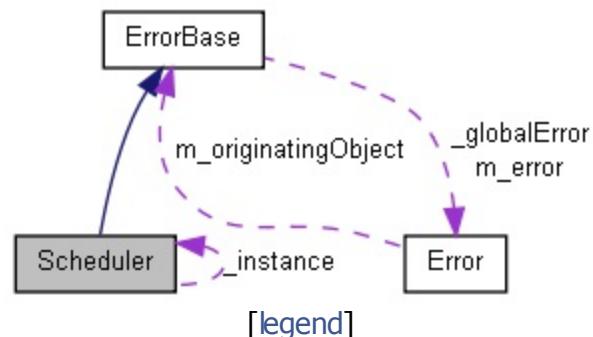
[Public Member Functions](#) |  
[Static Public Member Functions](#)

# Scheduler Class Reference

Inheritance diagram for Scheduler:



Collaboration diagram for Scheduler:



List of all members.

# Public Member Functions

void **AddCommand** (**Command** \*command)

Add a command to be scheduled later.

void **AddButton** (**ButtonScheduler** \*button)

void **RegisterSubsystem** (**Subsystem** \*subsystem)

Registers a **Subsystem** to this **Scheduler**, so that the **Scheduler** might know if a default **Command** needs to be run.

void **Run** ()

Runs a single iteration of the loop.

void **Remove** (**Command** \*command)

Removes the **Command** from the **Scheduler**.

void **RemoveAll** ()

void **SetEnabled** (bool enabled)

static **Scheduler** \* **GetInstance** ()

Returns the **Scheduler**, creating it if one does not exist.

# Member Function Documentation

## **void Scheduler::AddCommand ( Command \* command )**

Add a command to be scheduled later.

In any pass through the scheduler, all commands are added to the additions list, then at the end of the pass, they are all scheduled.

### **Parameters:**

**command** The command to be scheduled

## **Scheduler \* Scheduler::GetInstance ( ) [static]**

Returns the **Scheduler**, creating it if one does not exist.

### **Returns:**

the **Scheduler**

## **void Scheduler::RegisterSubsystem ( Subsystem \* subsystem )**

Registers a **Subsystem** to this **Scheduler**, so that the **Scheduler** might know if a default **Command** needs to be run.

All **Subsystems** should call this.

### **Parameters:**

**system** the system

## **void Scheduler::Remove ( Command \* command )**

Removes the **Command** from the **Scheduler**.

### **Parameters:**

**command** the command to remove

## **void Scheduler::Run ( )**

Runs a single iteration of the loop.

This method should be called often in order to have a functioning **Command** system.  
The loop has five stages:

1. Poll the Buttons
2. Execute/Remove the Commands
3. Send values to **SmartDashboard**
4. Add Commands
5. Add Defaults

---

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

- C:/WindRiver/workspace/WPILib/Commands/**Scheduler.h**
- C:/WindRiver/workspace/WPILib/Commands/Scheduler.cpp



# ScopedSocket Class Reference

---

Implements an object that automatically does a close on a camera socket on destruction. [More...](#)

[List of all members.](#)

## Public Member Functions

**ScopedSocket** (int camSock)

**operator int () const**

---

## Detailed Description

Implements an object that automatically does a close on a camera socket on destruction.

---

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

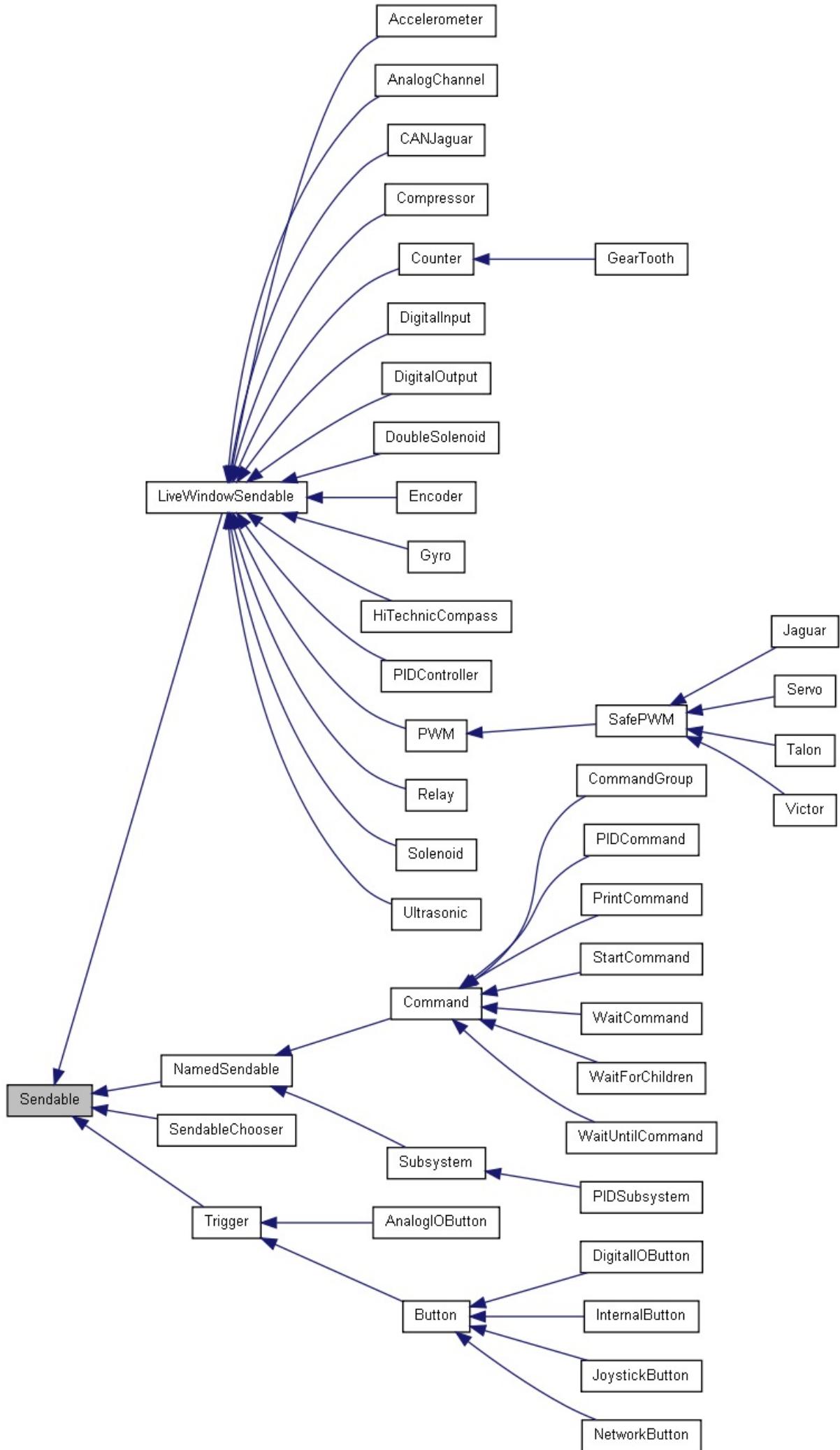
- C:/WindRiver/workspace/WPILib/Vision/PCVideoServer.cpp



# Sendable Class Reference

---

Inheritance diagram for Sendable:



## List of all members.

virtual void **InitTable (ITable \*subtable)=0**  
Initializes a table for this sendable object.

virtual **ITable \*** **GetTable ()=0**

virtual std::string **GetSmartDashboardType ()=0**

# Member Function Documentation

**virtual std::string Sendable::GetSmartDashboardType( )** [pure virtual]

## Returns:

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implemented in **Accelerometer**, **AnalogChannel**, **Trigger**, **CANJaguar**, **Command**, **PIDCommand**, **PIDSubsystem**, **Subsystem**, **Compressor**, **Counter**, **DigitalInput**, **DigitalOutput**, **DoubleSolenoid**, **Encoder**, **GearTooth**, **Gyro**, **HiTechnicCompass**, **PWM**, **Relay**, **Servo**, **SendableChooser**, **Solenoid**, and **Ultrasonic**.

**virtual ITable\* Sendable::GetTable( )** [pure virtual]

## Returns:

the table that is currently associated with the sendable

Implemented in **Accelerometer**, **AnalogChannel**, **Trigger**, **CANJaguar**, **Command**, **Subsystem**, **Compressor**, **Counter**, **DigitalInput**, **DigitalOutput**, **DoubleSolenoid**, **Encoder**, **Gyro**, **HiTechnicCompass**, **PWM**, **Relay**, **Servo**, **SendableChooser**, **Solenoid**, and **Ultrasonic**.

**virtual void Sendable::InitTable( ITable \* subtable )** [pure virtual]

Initializes a table for this sendable object.

## Parameters:

**subtable** The table to put the values in.

Implemented in **Accelerometer**, **AnalogChannel**, **Trigger**, **CANJaguar**, **Command**, **PIDCommand**, **PIDSubsystem**, **Subsystem**, **Compressor**, **Counter**, **DigitalInput**, **DigitalOutput**, **DoubleSolenoid**, **Encoder**, **Gyro**, **HiTechnicCompass**, **PIDController**, **PWM**, **Relay**, **Servo**, **SendableChooser**, **Solenoid**, and **Ultrasonic**.

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

- C:/WindRiver/workspace/WPILib/SmartDashboard/**Sendable.h**



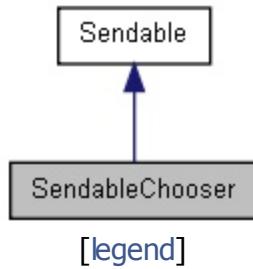


# SendableChooser Class Reference

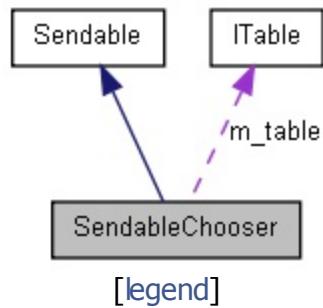
The **SendableChooser** class is a useful tool for presenting a selection of options to the **SmartDashboard**. More...

```
#include <SendableChooser.h>
```

Inheritance diagram for SendableChooser:



Collaboration diagram for SendableChooser:



List of all members.

## Public Member Functions

void **AddObject** (const char \*name, void \*object)

Adds the given object to the list of options.

void **AddDefault** (const char \*name, void \*object)

Add the given object to the list of options and marks it as the default.

void \* **GetSelected** ()

virtual void **InitTable** (**ITable** \*subtable)

Initializes a table for this sendable object.

virtual **ITable** \* **GetTable** ()

virtual std::string **GetSmartDashboardType** ()

## Detailed Description

The **SendableChooser** class is a useful tool for presenting a selection of options to the **SmartDashboard**.

For instance, you may wish to be able to select between multiple autonomous modes. You can do this by putting every possible **Command** you want to run as an autonomous into a **SendableChooser** and then put it into the **SmartDashboard** to have a list of options appear on the laptop. Once autonomous starts, simply ask the **SendableChooser** what the selected value is.

**See also:**

**SmartDashboard**

---

# Member Function Documentation

```
void SendableChooser::AddDefault( const char * name,  
                                 void *          object  
                               )
```

Add the given object to the list of options and marks it as the default.

Functionally, this is very close to [AddObject\(...\)](#) except that it will use this as the default option if none other is explicitly selected.

## Parameters:

**name** the name of the option  
**object** the option

```
void SendableChooser::AddObject( const char * name,  
                               void *          object  
                             )
```

Adds the given object to the list of options.

On the [SmartDashboard](#) on the desktop, the object will appear as the given name.

## Parameters:

**name** the name of the option  
**object** the option

```
virtual std::string SendableChooser::GetSmartDashboardType( ) [virtual]
```

## Returns:

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements [Sendable](#).

```
virtual ITable* SendableChooser::GetTable( ) [virtual]
```

## Returns:

the table that is currently associated with the sendable

Implements **Sendable**.

## **virtual void SendableChooser::InitTable( ITable \* **subtable** ) [virtual]**

Initializes a table for this sendable object.

### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

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

- C:/WindRiver/workspace/WPILib/SmartDashboard/**SendableChooser.h**
- C:/WindRiver/workspace/WPILib/SmartDashboard/SendableChooser.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

[Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Static Public Attributes](#) |  
[Protected Member Functions](#)

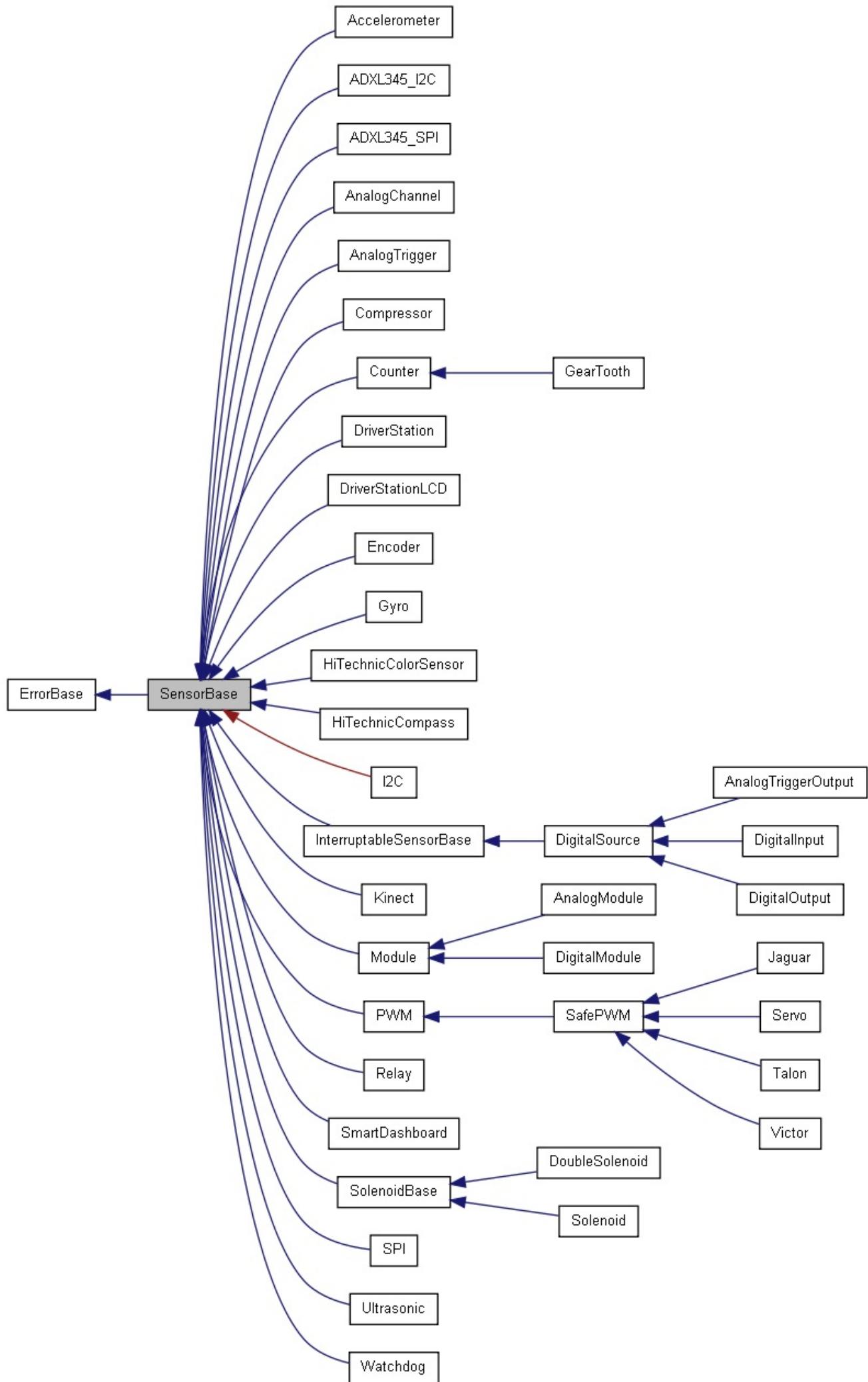
# SensorBase Class Reference

---

Base class for all sensors. [More...](#)

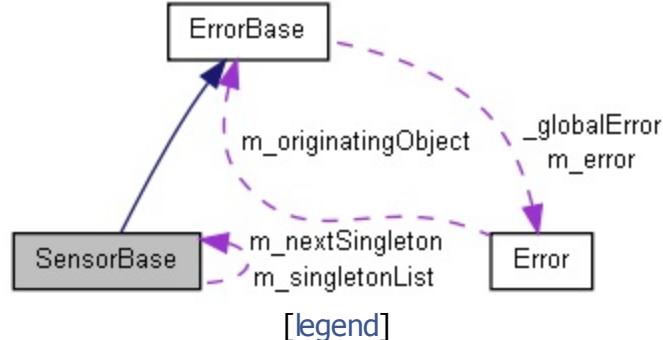
```
#include <SensorBase.h>
```

Inheritance diagram for SensorBase:



[legend]

## Collaboration diagram for SensorBase:



List of all members.

### **SensorBase ()**

Creates an instance of the sensor base and gets an FPGA handle.

virtual **~SensorBase ()**

Frees the resources for a **SensorBase**.

static void **DeleteSingletons ()**  
Delete all the singleton classes on the list.

static UINT32 **GetDefaultAnalogModule ()**

static UINT32 **GetDefaultDigitalModule ()**

static UINT32 **GetDefaultSolenoidModule ()**

static bool **CheckAnalogModule** (UINT8 moduleNumber)  
Check that the analog module number is valid.

static bool **CheckDigitalModule** (UINT8 moduleNumber)  
Check that the digital module number is valid.

static bool **CheckPWMModule** (UINT8 moduleNumber)  
Check that the digital module number is valid.

static bool **CheckRelayModule** (UINT8 moduleNumber)  
Check that the digital module number is valid.

static bool **CheckSolenoidModule** (UINT8 moduleNumber)  
Check that the solenoid module number is valid.

static bool **CheckDigitalChannel** (UINT32 channel)  
Check that the digital channel number is valid.

static bool **CheckRelayChannel** (UINT32 channel)  
Check that the digital channel number is valid.

static bool **CheckPWMChannel** (UINT32 channel)  
Check that the digital channel number is valid.

static bool **CheckAnalogChannel** (UINT32 channel)  
Check that the analog channel number is value.

static bool **CheckSolenoidChannel** (UINT32 channel)  
Verify that the solenoid channel number is within limits.

## Static Public Attributes

```
static const UINT32 kSystemClockTicksPerMicrosecond = 40
static const UINT32 kDigitalChannels = 14
static const UINT32 kAnalogChannels = 8
static const UINT32 kAnalogModules = 2
static const UINT32 kDigitalModules = 2
static const UINT32 kSolenoidChannels = 8
static const UINT32 kSolenoidModules = 2
static const UINT32 kPwmChannels = 10
static const UINT32 kRelayChannels = 8
static const UINT32 kChassisSlots = 8
```

void **AddToSingletonList ()**

Add sensor to the singleton list.

## Detailed Description

Base class for all sensors.

Stores most recent status information as well as containing utility functions for checking channels and error processing.

---

# Member Function Documentation

## **void SensorBase::AddToSingletonList( )** [protected]

Add sensor to the singleton list.

Add this sensor to the list of singletons that need to be deleted when the robot program exits. Each of the sensors on this list are singletons, that is they aren't allocated directly with new, but instead are allocated by the static GetInstance method. As a result, they are never deleted when the program exits. Consequently these sensors may still be holding onto resources and need to have their destructors called at the end of the program.

## **bool SensorBase::CheckAnalogChannel( UINT32 channel )** [static]

Check that the analog channel number is valid.

Verify that the analog channel number is one of the legal channel numbers. Channel numbers are 1-based.

### **Returns:**

Analog channel is valid

## **bool SensorBase::CheckAnalogModule( UINT8 moduleNumber )** [static]

Check that the analog module number is valid.

### **Returns:**

Analog module is valid and present

## **bool SensorBase::CheckDigitalChannel( UINT32 channel )** [static]

Check that the digital channel number is valid.

Verify that the channel number is one of the legal channel numbers. Channel numbers are 1-based.

### **Returns:**

Digital channel is valid

## **bool SensorBase::CheckDigitalModule ( **UINT8 moduleNumber** ) [static]**

Check that the digital module number is valid.

### **Returns:**

Digital module is valid and present

## **bool SensorBase::CheckPWMChannel ( **UINT32 channel** ) [static]**

Check that the digital channel number is valid.

Verify that the channel number is one of the legal channel numbers. Channel numbers are 1-based.

### **Returns:**

PWM channel is valid

## **bool SensorBase::CheckPWMModule ( **UINT8 moduleNumber** ) [static]**

Check that the digital module number is valid.

### **Returns:**

Digital module is valid and present

## **bool SensorBase::CheckRelayChannel ( **UINT32 channel** ) [static]**

Check that the digital channel number is valid.

Verify that the channel number is one of the legal channel numbers. Channel numbers are 1-based.

### **Returns:**

Relay channel is valid

## **bool SensorBase::CheckRelayModule ( **UINT8 moduleNumber** ) [static]**

Check that the digital module number is valid.

### **Returns:**

Digital module is valid and present

## **bool SensorBase::CheckSolenoidChannel ( **UINT32 channel** ) [static]**

Verify that the solenoid channel number is within limits.

### **Returns:**

**Solenoid** channel is valid

## **bool SensorBase::CheckSolenoidModule ( **UINT8 moduleNumber** ) [static]**

Check that the solenoid module number is valid.

### **Returns:**

**Solenoid** module is valid and present

## **void SensorBase::DeleteSingletons ( ) [static]**

Delete all the singleton classes on the list.

All the classes that were allocated as singletons need to be deleted so their resources can be freed.

---

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

- C:/WindRiver/workspace/WPILib/**SensorBase.h**
- C:/WindRiver/workspace/WPILib/SensorBase.cpp

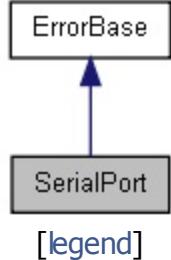


# SerialPort Class Reference

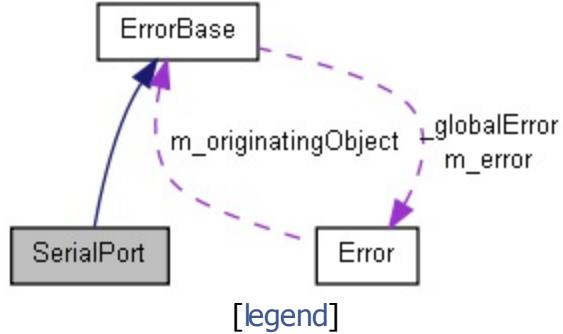
Driver for the RS-232 serial port on the cRIO. [More...](#)

```
#include <SerialPort.h>
```

Inheritance diagram for SerialPort:



Collaboration diagram for SerialPort:



[List of all members.](#)

# Public Types

```
Parity {  
    kParity_None = 0, kParity_Odd = 1, kParity_Even = 2, kParity_Mark  
enum = 3,  
    kParity_Space = 4  
}  
  
enum StopBits { kStopBits_One = 10, kStopBits_OnePointFive = 15,  
kStopBits_Two = 20 }  
  
enum FlowControl { kFlowControl_None = 0, kFlowControl_XonXoff = 1,  
kFlowControl_RtsCts = 2, kFlowControl_DtrDsr = 4 }  
enum WriteBufferMode { kFlushOnAccess = 1, kFlushWhenFull = 2 }
```

**SerialPort** (UINT32 baudRate, UINT8 dataBits=8, Parity parity=kParity\_None, StopBits stopBits=kStopBits\_One)  
Create an instance of a Serial Port class.

**~SerialPort ()**

Destructor.

**void SetFlowControl (FlowControl flowControl)**

Set the type of flow control to enable on this port.

**void EnableTermination (char terminator= '\n')**

Enable termination and specify the termination character.

**void DisableTermination ()**

Disable termination behavior.

**INT32 GetBytesReceived ()**

Get the number of bytes currently available to read from the serial port.

**void Printf (const char \*writeFmt,...)**

Output formatted text to the serial port.

**void Scanf (const char \*readFmt,...)**

Input formatted text from the serial port.

**UINT32 Read (char \*buffer, INT32 count)**

Read raw bytes out of the buffer.

UINT32 **Write** (const char \*buffer, INT32 count)  
Write raw bytes to the buffer.

void **SetTimeout** (float timeout)  
Configure the timeout of the serial port.

void **SetReadBufferSize** (UINT32 size)  
Specify the size of the input buffer.

void **SetWriteBufferSize** (UINT32 size)  
Specify the size of the output buffer.

void **SetWriteBufferMode** (WriteBufferMode mode)  
Specify the flushing behavior of the output buffer.

void **Flush** ()  
Force the output buffer to be written to the port.

void **Reset** ()  
Reset the serial port driver to a known state.

## Detailed Description

Driver for the RS-232 serial port on the cRIO.

The current implementation uses the VISA formatted I/O mode. This means that all traffic goes through the fomatted buffers. This allows the intermingled use of **Printf()**, **Scanf()**, and the raw buffer accessors **Read()** and **Write()**.

More information can be found in the NI-VISA User Manual here:

<http://www.ni.com/pdf/manuals/370423a.pdf> and the NI-VISA Programmer's Reference Manual here: <http://www.ni.com/pdf/manuals/370132c.pdf>

---

# Constructor & Destructor Documentation

```
SerialPort::SerialPort( UINT32
                        UINT8
                        SerialPort::Parity     baudRate,
                        SerialPort::StopBits   dataBits = 8,
                                              parity = kParity_None,
                                              stopBits = kStopBits_One
                    )
```

Create an instance of a Serial Port class.

## 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.

# Member Function Documentation

```
void SerialPort::EnableTermination ( char terminator = '\n' )
```

Enable termination and specify the termination character.

Termination is currently only implemented for receive. When the the terminator is received, the **Read()** or **Scanf()** will return fewer bytes than requested, stopping after the terminator.

## Parameters:

**terminator** The character to use for termination.

## **void SerialPort::Flush( )**

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.

## **INT32 SerialPort::GetBytesReceived ( )**

Get the number of bytes currently available to read from the serial port.

## Returns:

The number of bytes available to read.

```
void SerialPort::Printf ( const char * writeFmt,
```

1

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.

```
UINT32 SerialPort::Read ( 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.

```
void SerialPort::Reset ( )
```

Reset the serial port driver to a known state.

Empty the transmit and receive buffers in the device and formatted I/O.

```
void SerialPort::Scanf ( 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.

```
void SerialPort::SetFlowControl ( SerialPort::FlowControl flowControl )
```

Set the type of flow control to enable on this port.

By default, flow control is disabled.

```
void SerialPort::SetReadBufferSize ( UINT32 size )
```

Specify the size of the input buffer.

Specify the amount of data that can be stored before data from the device is returned to Read or Scanf. If you want data that is received to be returned immediately, set this to 1.

If the buffer is not filled before the read timeout expires, all data that has been received so far will be returned.

#### Parameters:

**size** The read buffer size.

### **void SerialPort::SetTimeout ( float 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 wait for I/O.

### **void SerialPort::SetWriteBufferMode ( 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.

### **void SerialPort::SetWriteBufferSize ( UINT32 size )**

Specify the size of the output buffer.

Specify the amount of data that can be stored before being transmitted to the device.

## Parameters:

**size** The write buffer size.

```
UINT32 SerialPort::Write ( 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.

---

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

- C:/WindRiver/workspace/WPILib/[SerialPort.h](#)
- C:/WindRiver/workspace/WPILib/SerialPort.cpp

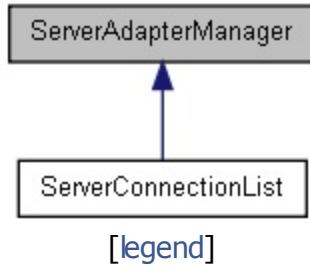


# ServerAdapterManager Class Reference

A class that manages connections to a server. [More...](#)

```
#include <ServerAdapterManager.h>
```

Inheritance diagram for ServerAdapterManager:



[List of all members.](#)

## Public Member Functions

virtual void **close** ([ServerConnectionAdapter](#) &connectionAdapter, bool closeStream)=0  
Called when a connection adapter has been closed.

---

# Detailed Description

A class that manages connections to a server.

## Author:

Mitchell

---

# Member Function Documentation

```
virtual void ServerAdapterManager::close ( ServerConnectionAdapter & conn  
                                         bool close  
                                         ) [pure]
```

Called when a connection adapter has been closed.

## Parameters:

**connectionAdapter** the adapter that was closed

Implemented in **ServerConnectionList**.

---

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

- C:/WindRiver/workspace/WPILib/networktables2/server/**ServerAdapterManager.h**

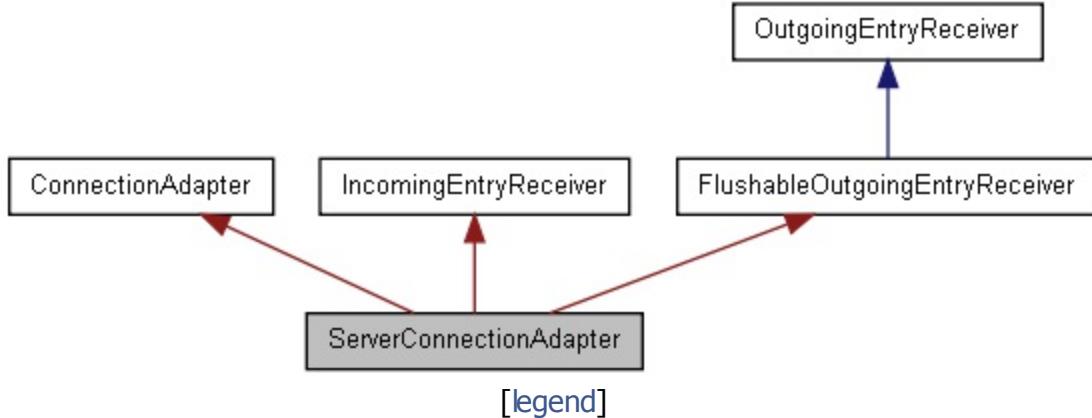


# ServerConnectionAdapter Class Reference

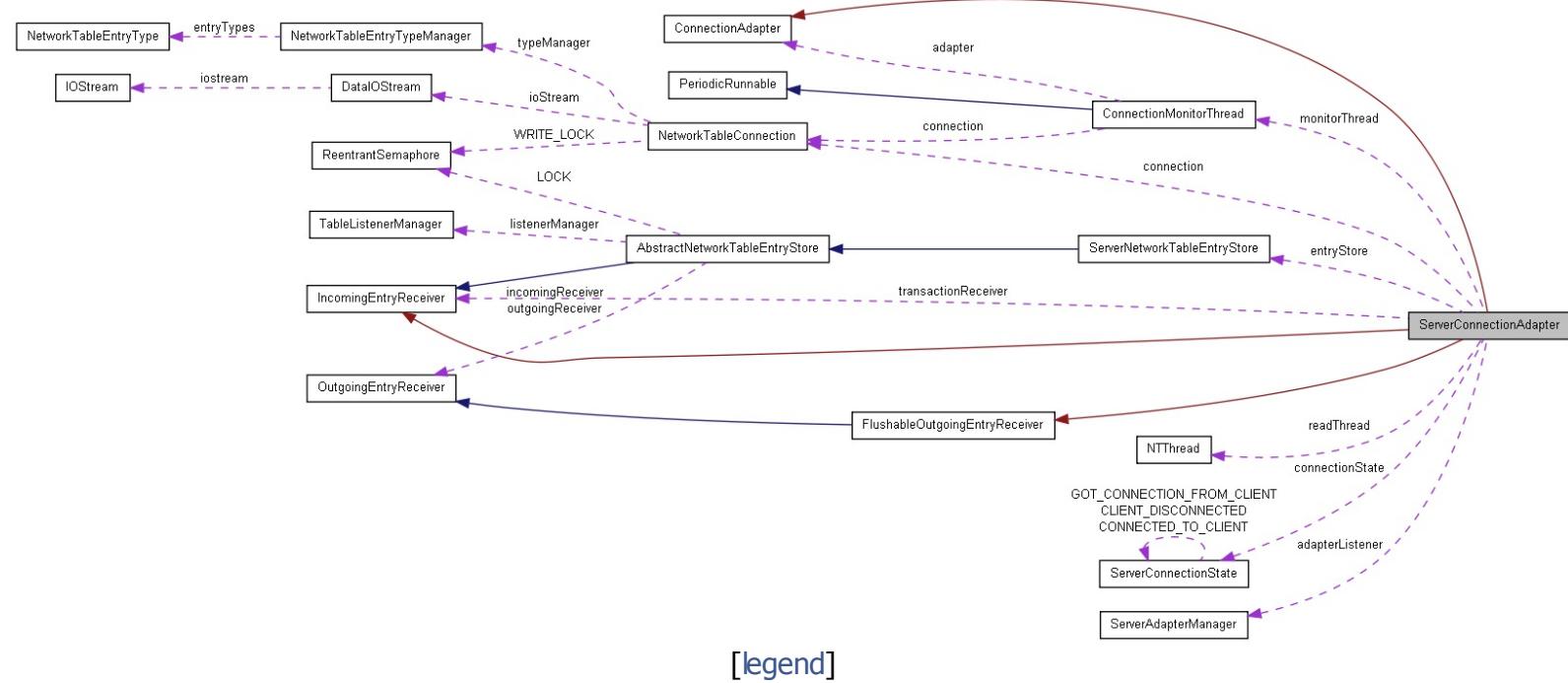
Object that adapts messages from a client to the server. [More...](#)

```
#include <ServerConnectionAdapter.h>
```

Inheritance diagram for ServerConnectionAdapter:



Collaboration diagram for ServerConnectionAdapter:



List of all members.

# Public Member Functions

**ServerConnectionAdapter** (**IOStream** \*stream,  
**ServerNetworkTableEntryStore** &entryStore,  
**IncomingEntryReceiver** &transactionReceiver,  
**ServerAdapterManager** &adapterListener,  
**NetworkTableEntryTypeManager** &typeManager,  
**NTThreadManager** &threadManager)

Create a server connection adapter for a given stream.

void **badMessage** (**BadMessageException** &e)  
called if a bad message exception is thrown

void **ioException** (**IOException** &e)  
called if an io exception is thrown

void **shutdown** (bool closeStream)  
stop the read thread and close the stream

void **keepAlive** ()

void **clientHello** (ProtocolVersion protocolRevision)

void **protocolVersionUnsupported** (ProtocolVersion protocolRevision)

void **serverHelloComplete** ()

void **offerIncomingAssignment** (**NetworkTableEntry** \*entry)

void **offerIncomingUpdate** (**NetworkTableEntry** \*entry,  
SequenceNumber sequenceNumber, **EntryValue** value)

**NetworkTableEntry** \* **GetEntry** (EntryId id)

void **offerOutgoingAssignment** (**NetworkTableEntry** \*entry)

void **offerOutgoingUpdate** (**NetworkTableEntry** \*entry)

void **flush** ()

**ServerConnectionState** \* **getConnectionState** ()

void **ensureAlive** ()

## Public Attributes

**NetworkTableConnection connection**

the connection this adapter uses

## **Detailed Description**

Object that adapts messages from a client to the server.

### **Author:**

Mitchell

---

# Constructor & Destructor Documentation

**ServerConnectionAdapter::ServerConnectionAdapter (**[IOStream \\*](#)  
[ServerNetworkTableEn](#)  
[IncomingEntryReceive](#)  
[ServerAdapterManager](#)  
[NetworkTableEntryTyp](#)  
[NTThreadManager](#) &  
**)**

Create a server connection adapter for a given stream.

## Parameters:

**stream**  
**transactionPool**  
**entryStore**  
**transactionReceiver**  
**adapterListener**  
**threadManager**

# Member Function Documentation

## **void ServerConnectionAdapter::badMessage ( BadMessageException & e )** [virtual]

called if a bad message exception is thrown

### Parameters:

e

Implements **ConnectionAdapter**.

## **ServerConnectionState \* ServerConnectionAdapter::get ConnectionState ( )**

### Returns:

the state of the connection

## **void ServerConnectionAdapter::ioException ( IOException & e )** [virtual]

called if an io exception is thrown

### Parameters:

e

Implements **ConnectionAdapter**.

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

- C:/WindRiver/workspace/WPILib/networktables2/server/**ServerConnectionAdapter.h**
- C:/WindRiver/workspace/WPILib/networktables2/server/ServerConnectionAdapter.cpp

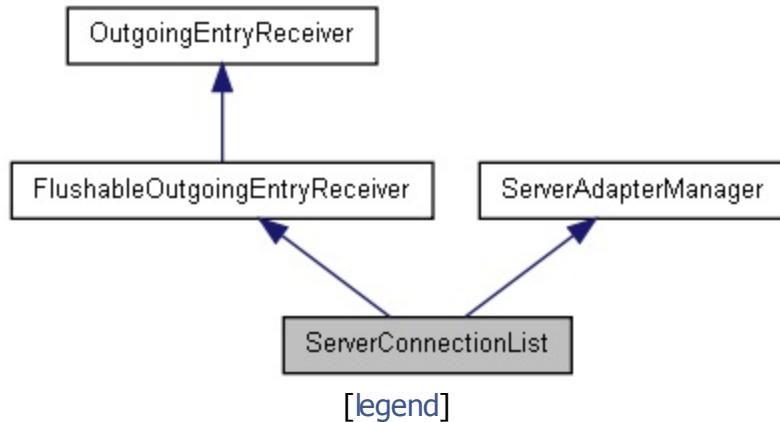


# ServerConnectionList Class Reference

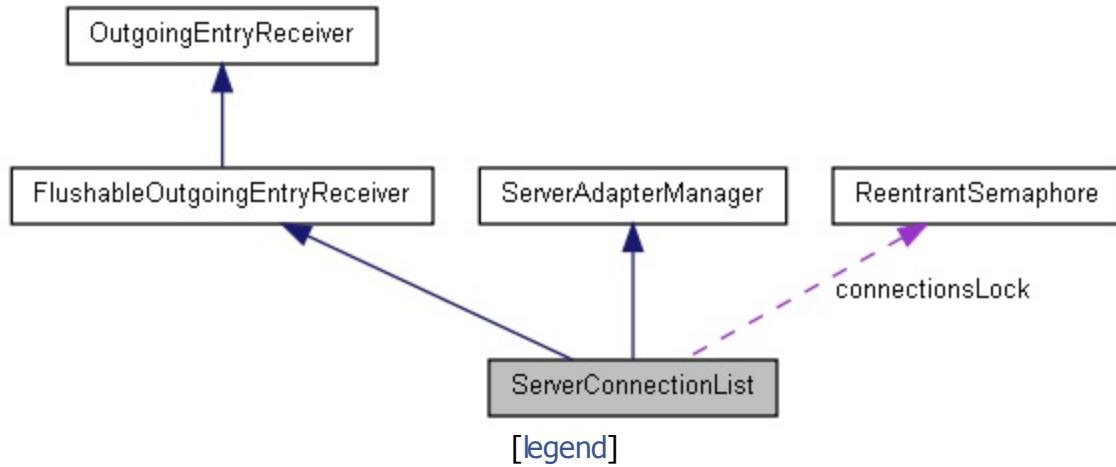
A list of connections that the server currently has. [More...](#)

```
#include <ServerConnectionList.h>
```

Inheritance diagram for ServerConnectionList:



Collaboration diagram for ServerConnectionList:



[List of all members.](#)

## Public Member Functions

void **add** (**ServerConnectionAdapter** &connection)

Add a connection to the list.

void **close** (**ServerConnectionAdapter** &connectionAdapter, bool closeStream)

Called when a connection adapter has been closed.

void **closeAll** ()

close all connections and remove them

void **offerOutgoingAssignment** (**NetworkTableEntry** \*entry)

void **offerOutgoingUpdate** (**NetworkTableEntry** \*entry)

void **flush** ()

void **ensureAlive** ()

## Detailed Description

A list of connections that the server currently has.

### **Author:**

Mitchell

---

# Member Function Documentation

## **void ServerConnectionList::add ( [ServerConnectionAdapter](#) & [connection](#) )**

Add a connection to the list.

### Parameters:

**connection**

## **void ServerConnectionList::close ( [ServerConnectionAdapter](#) & [connectionAdapter](#),   bool [closeStream](#),   )**

**[virtual]**

Called when a connection adapter has been closed.

### Parameters:

**connectionAdapter** the adapter that was closed

Implements [ServerAdapterManager](#).

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

- C:/WindRiver/workspace/WPILib/networktables2/server/[ServerConnectionList.h](#)
- C:/WindRiver/workspace/WPILib/networktables2/server/ServerConnectionList.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

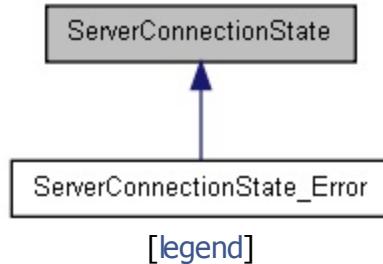
[Public Member Functions](#) |  
[Static Public Attributes](#) |  
[Protected Member Functions](#)

# ServerConnectionState Class Reference

Represents the state of a connection to the server. [More...](#)

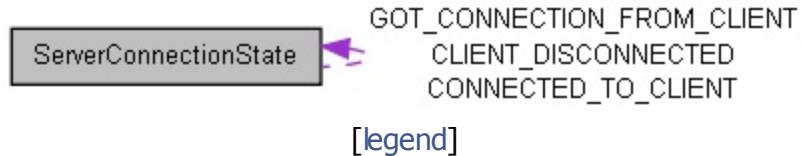
```
#include <ServerConnectionState.h>
```

Inheritance diagram for ServerConnectionState:



[[legend](#)]

Collaboration diagram for ServerConnectionState:



[[legend](#)]

[List of all members.](#)

# Public Member Functions

```
virtual const char * toString ()
```

static <b>ServerConnectionState</b>	<b>GOT_CONNECTION_FROM_CLIENT</b>	represents that the server has received the connection from the client but has not yet received the client hello
static <b>ServerConnectionState</b>	<b>CONNECTED_TO_CLIENT</b>	represents that the client is in a connected non-error state
static <b>ServerConnectionState</b>	<b>CLIENT_DISCONNECTED</b>	represents that the client has disconnected from the server



# Detailed Description

Represents the state of a connection to the server.

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/networktables2/server/**ServerConnectionState.h**
- C:/WindRiver/workspace/WPILib/networktables2/server/ServerConnectionState.cpp

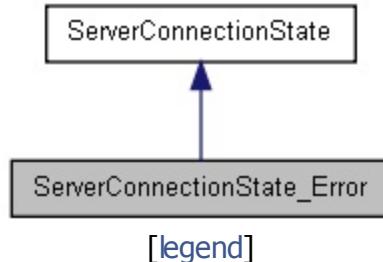


# ServerConnectionState\_Error Class Reference

Represents that the client is in an error state. [More...](#)

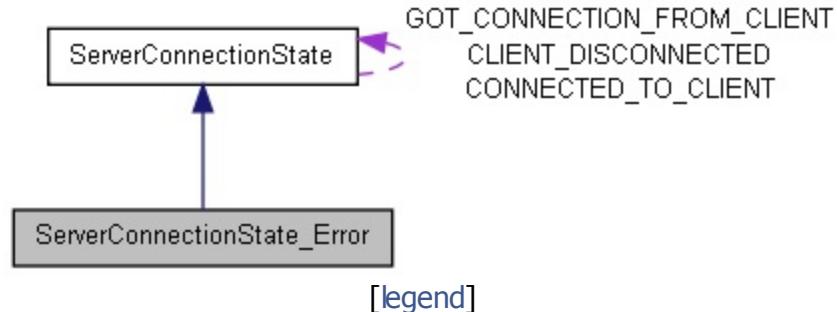
```
#include <ServerConnectionState.h>
```

Inheritance diagram for ServerConnectionState\_Error:



[\[legend\]](#)

Collaboration diagram for ServerConnectionState\_Error:



[\[legend\]](#)

[List of all members.](#)

## Public Member Functions

**ServerConnectionState\_Error** (std::exception &e)

Create a new error state.

virtual const char \* **toString** ()

std::exception & **getException** ()

---

## Detailed Description

Represents that the client is in an error state.

### **Author:**

Mitchell

---

# Constructor & Destructor Documentation

## **ServerConnectionState\_Error::ServerConnectionState\_Error ( std::exception &**

Create a new error state.

### **Parameters:**

**e**

---

# Member Function Documentation

## **std::exception & ServerConnectionState\_Error::getException( )**

### Returns:

the exception that caused the client connection to enter an error state

---

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

- C:/WindRiver/workspace/WPILib/networktables2/server/**ServerConnectionState.h**
- C:/WindRiver/workspace/WPILib/networktables2/server/ServerConnectionState.cpp

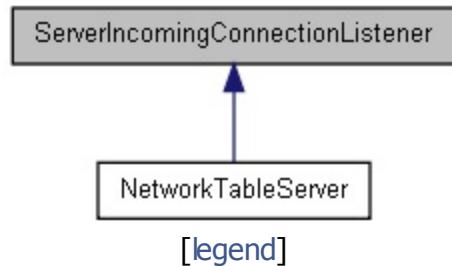


# ServerIncomingConnectionListener Class Reference

Listener for new incoming server connections. [More...](#)

```
#include <ServerIncomingConnectionListener.h>
```

Inheritance diagram for ServerIncomingConnectionListener:



[[legend](#)]

[List of all members.](#)

## Public Member Functions

virtual void **OnNewConnection** (**ServerConnectionAdapter**  
&connectionAdapter)=0  
Called on create of a new connection.

---

## Detailed Description

Listener for new incoming server connections.

### Author:

Mitchell

---

# Member Function Documentation

## **virtual void ServerIncomingConnectionListener::OnNewConnection ( ServerCo**

Called on create of a new connection.

### Parameters:

**connectionAdapter** the server connection adapter

Implemented in **NetworkTableServer**.

---

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

- C:/WindRiver/workspace/WPILib/networktables2/server/**ServerIncomingConnecti**

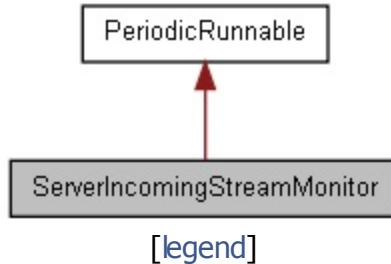


# ServerIncomingStreamMonitor Class Reference

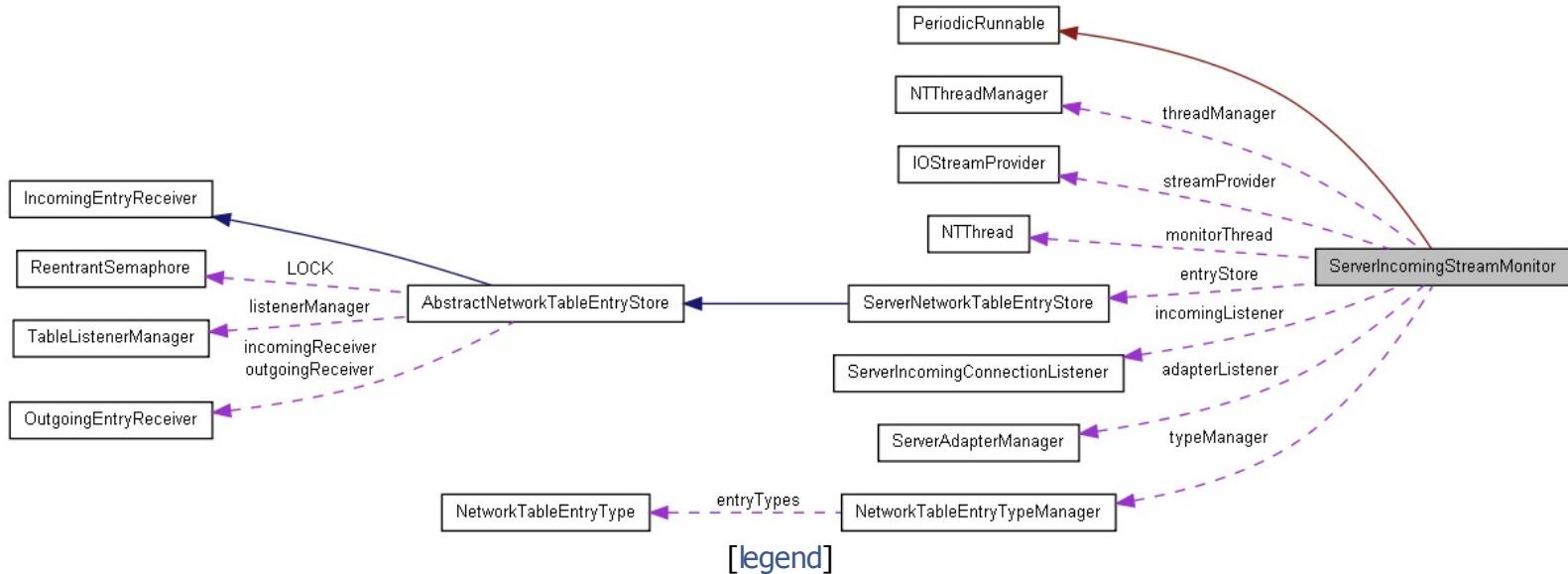
Thread that monitors for incoming connections. More...

```
#include <ServerIncomingStreamMonitor.h>
```

Inheritance diagram for ServerIncomingStreamMonitor:



Collaboration diagram for ServerIncomingStreamMonitor:



List of all members.

**ServerIncomingStreamMonitor** (**IOStreamProvider** &streamProvider,  
**ServerNetworkTableEntryStore** &entryStore,  
**ServerIncomingConnectionListener** &incomingListener,

**ServerAdapterManager** &adapterListener,  
**NetworkTableEntryTypeManager** &typeManager, **NTThreadManager**  
&threadManager)

Create a new incoming stream monitor.

void **start ()**

Start the monitor thread.

void **stop ()**

Stop the monitor thread.

void **run ()**

the method that will be called periodically on a thread

## Detailed Description

Thread that monitors for incoming connections.

### **Author:**

Mitchell

---

# Constructor & Destructor Documentation

**ServerIncomingStreamMonitor::ServerIncomingStreamMonitor ( [IOStreamProvider](#), [ServerNetworkTable](#), [ServerIncomingStreamMonitor](#), [ServerAdapter](#), [NetworkTable](#), [NTThreadManager](#) )**

Create a new incoming stream monitor.

## Parameters:

<b>streamProvider</b>	the stream provider to retrieve streams from
<b>entryStore</b>	the entry store for the server
<b>transactionPool</b>	transaction pool for the server
<b>incomingListener</b>	the listener that is notified of new connections
<b>adapterListener</b>	the listener that will listen to adapter events
<b>threadManager</b>	the thread manager used to create the incoming thread and provided to the Connection Adapters

# Member Function Documentation

## **void ServerIncomingStreamMonitor::run( ) [virtual]**

the method that will be called periodically on a thread

### **Exceptions:**

thrown when the thread is supposed to be interrupted and **InterruptedException** stop (implementers should always let this exception fall through)

Implements **PeriodicRunnable**.

---

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

- C:/WindRiver/workspace/WPILib/networktables2/server/**ServerIncomingStreamM**
- C:/WindRiver/workspace/WPILib/networktables2/server/ServerIncomingStreamMonit

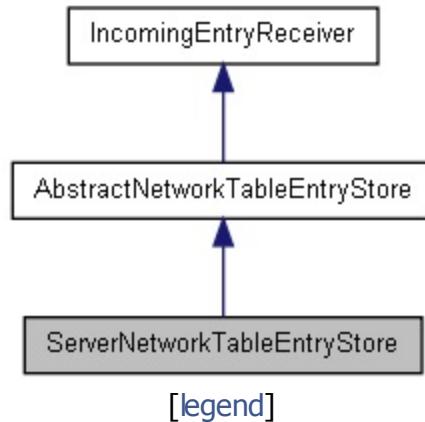
[Public Member Functions](#) |  
[Protected Member Functions](#)

# ServerNetworkTableEntryStore Class Reference

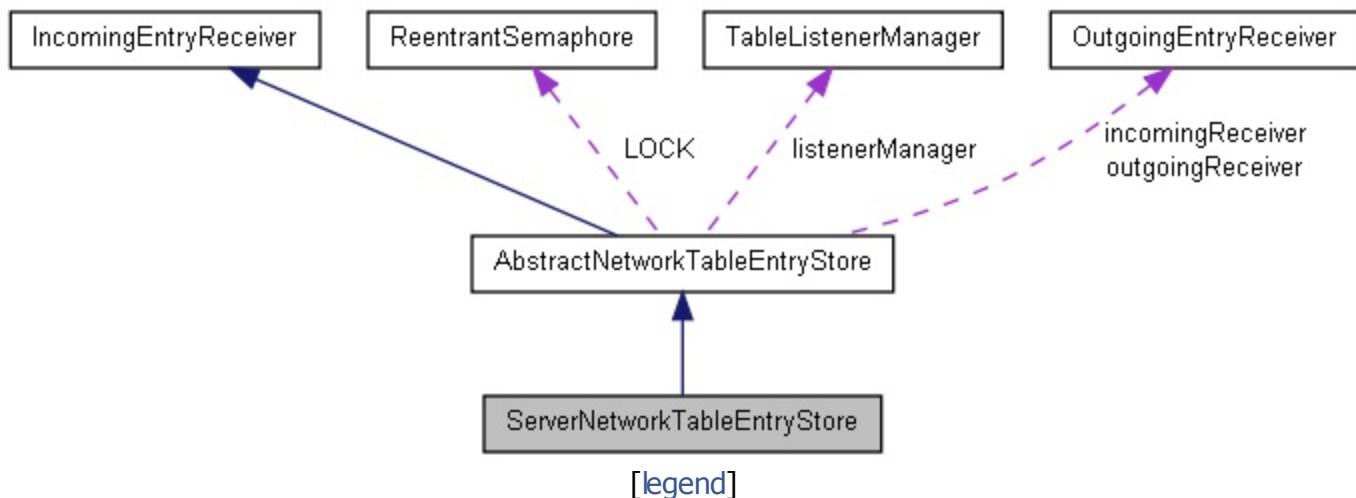
The entry store for a **NetworkTableServer**. More...

```
#include <ServerNetworkTableEntryStore.h>
```

Inheritance diagram for ServerNetworkTableEntryStore:



Collaboration diagram for ServerNetworkTableEntryStore:



List of all members.

# Public Member Functions

**ServerNetworkTableEntryStore (TableListenerManager &listenerManager)**

Create a new Server entry store.

**void sendServerHello (NetworkTableConnection &connection)**

Send all entries in the entry store as entry assignments in a single transaction.

---

**bool addEntry (NetworkTableEntry \*newEntry)**

**updateEntry (NetworkTableEntry \*entry, SequenceNumber**

**bool sequenceNumber, EntryValue value)**

---

## Detailed Description

The entry store for a [NetworkTableServer](#).

### Author:

Mitchell

---

# Constructor & Destructor Documentation

## **ServerNetworkTableEntryStore::ServerNetworkTableEntryStore ( TableListener< T> listenerManager )**

Create a new Server entry store.

### Parameters:

**transactionPool** the transaction pool

**listenerManager** the listener manager that fires events from this entry store

# Member Function Documentation

## **void ServerNetworkTableEntryStore::sendServerHello ( [NetworkTableConnect](#)**

Send all entries in the entry store as entry assignments in a single transaction.

### **Parameters:**

**connection**

### **Exceptions:**

**[IOException](#)**

---

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

- C:/WindRiver/workspace/WPILib/networktables2/server/**ServerNetworkTableEntr**
- C:/WindRiver/workspace/WPILib/networktables2/server/ServerNetworkTableEntrySto

[Class List](#)[Class Hierarchy](#)[Class Members](#)

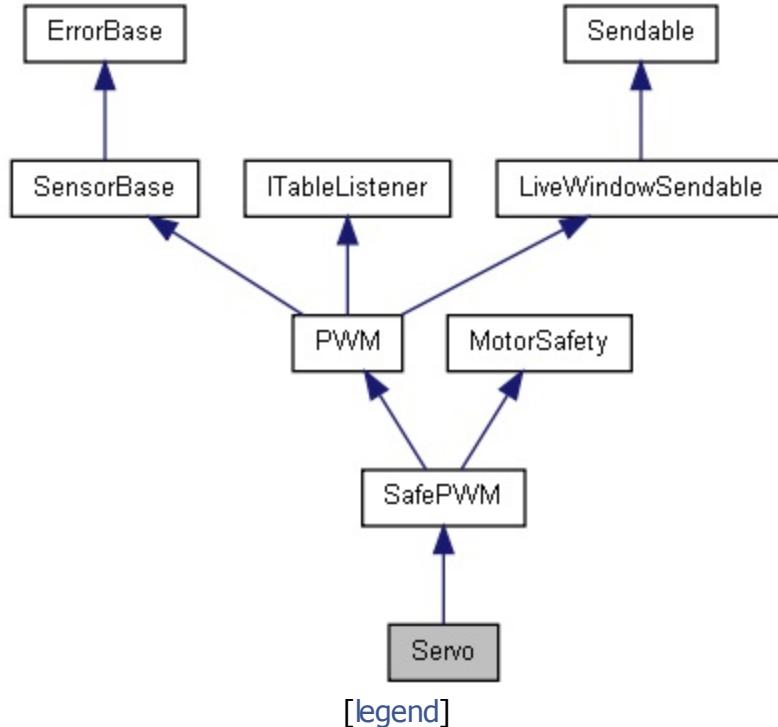
[Public Member Functions](#) |  
[Static Public Member Functions](#) |  
[Public Attributes](#)

# Servo Class Reference

Standard hobby style servo. More...

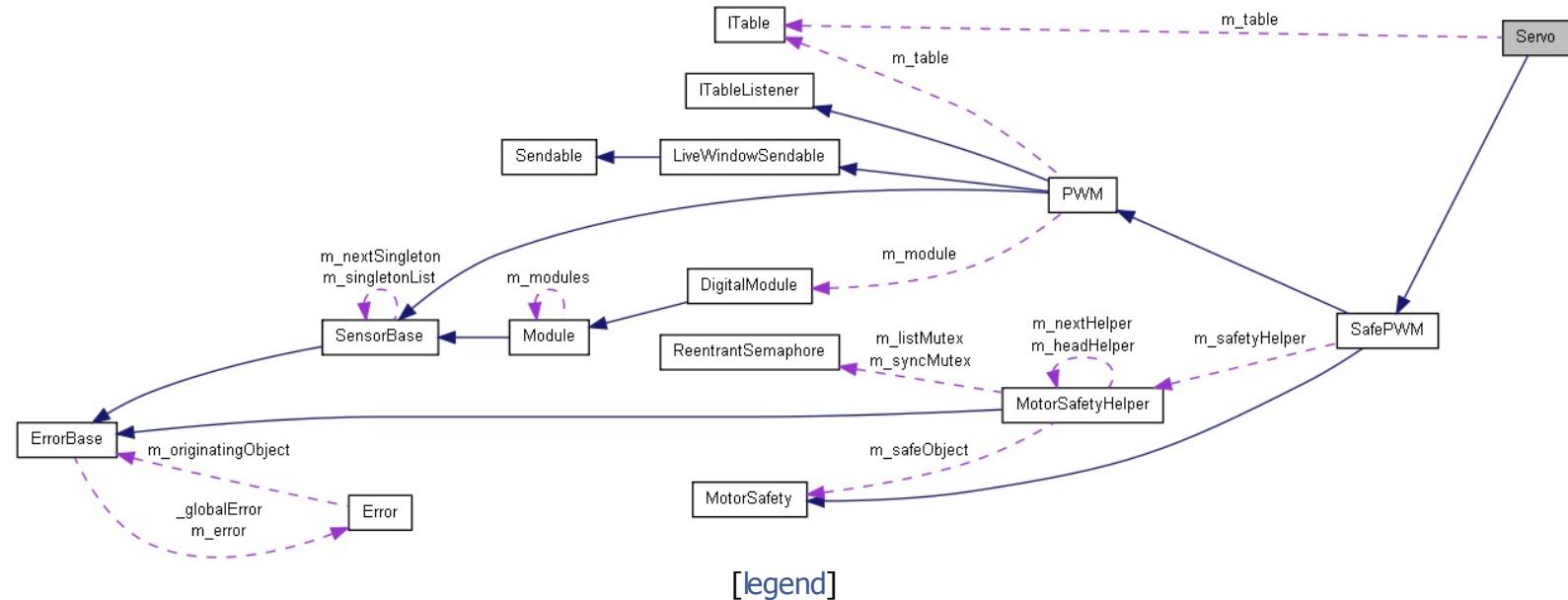
```
#include <Servo.h>
```

Inheritance diagram for Servo:



[legend]

Collaboration diagram for Servo:



[legend]

List of all members.

# Public Member Functions

## **Servo** (UINT32 channel)

Constructor that assumes the default digital module.

## **Servo** (UINT8 moduleNumber, UINT32 channel)

Constructor that specifies the digital module.

void **Set**(float value)  
Set the servo position.

void **SetOffline** ()  
Set the servo to offline.

float **Get** ()  
Get the servo position.

void **SetAngle** (float angle)  
Set the servo angle.

float **GetAngle** ()  
Get the servo angle.

void **ValueChanged** (ITable \*source, const std::string &key, EntryValue  
value, bool isNew)

Called when a key-value pair is changed in a **ITable** WARNING: If a new  
key-value is put in this method value changed will immediately be called  
which could lead to recursive code.

void **UpdateTable** ()  
Update the table for this sendable object with the latest values.

void **StartLiveWindowMode** ()  
Start having this sendable object automatically respond to value changes  
reflect the value on the table.

void **StopLiveWindowMode** ()  
Stop having this sendable object automatically respond to value changes.

std::string **GetSmartDashboardType** ()

void **InitTable** (ITable \*subTable)  
Initializes a table for this sendable object.

**ITable** \* **GetTable** ()

static float **GetMaxAngle** ()

static float **GetMinAngle** ()

## Public Attributes

**ITable \* m\_table**

## Detailed Description

Standard hobby style servo.

The range parameters default to the appropriate values for the Hitec HS-322HD servo provided in the FIRST Kit of Parts in 2008.

---

# Constructor & Destructor Documentation

## Servo::Servo ( **UINT32 channel** ) [explicit]

Constructor that assumes the default digital module.

### Parameters:

**channel** The **PWM** channel on the digital module to which the servo is attached.

## Servo::Servo ( **UINT8 moduleNumber,** **UINT32 channel** )

Constructor that specifies the digital module.

### Parameters:

**moduleNumber** The digital module (1 or 2).

**channel** The **PWM** channel on the digital module to which the servo is attached (1..10).

# Member Function Documentation

## **float Servo::Get( )**

Get the servo position.

**Servo** values range from 0.0 to 1.0 corresponding to the range of full left to full right.

### **Returns:**

Position from 0.0 to 1.0.

## **float Servo::GetAngle ( void )**

Get the servo angle.

Assume that the servo angle is linear with respect to the **PWM** value (big assumption, need to test).

### **Returns:**

The angle in degrees to which the servo is set.

## **std::string Servo::GetSmartDashboardType ( ) [virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Reimplemented from **PWM**.

## **ITable \* Servo::GetTable ( ) [virtual]**

### **Returns:**

the table that is currently associated with the sendable

Reimplemented from **PWM**.

## **void Servo::InitTable ( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

## Parameters:

**subtable** The table to put the values in.

Reimplemented from **PWM**.

## **void Servo::Set( 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:

**value** Position from 0.0 to 1.0.

## **void Servo::SetAngle( float degrees )**

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:

**degrees** The angle in degrees to set the servo.

## **void Servo::SetOffline( )**

Set the servo to offline.

Set the servo raw value to 0 (undriven)

## **void Servo::ValueChanged( ITable \* source, const std::string & key, EntryValue value, bool isNew )**

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

### Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Reimplemented from **PWM**.

---

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

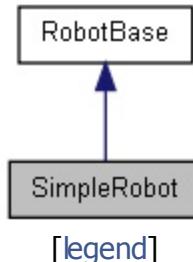
- C:/WindRiver/workspace/WPILib/**Servo.h**
- C:/WindRiver/workspace/WPILib/Servo.cpp



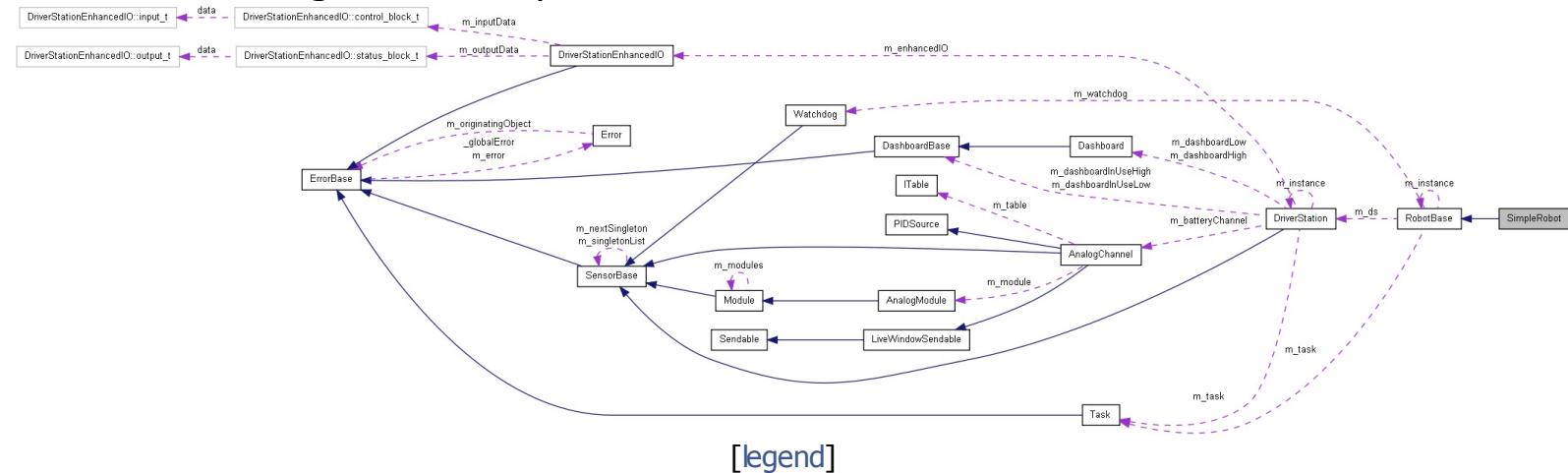
# SimpleRobot Class Reference

```
#include <SimpleRobot.h>
```

## Inheritance diagram for SimpleRobot:



## Collaboration diagram for SimpleRobot:



## List of all members.

## virtual void **RobotInit** ()

Robot-wide initialization code should go here.

## virtual void **Disabled** ()

Disabled should go here.

virtual void **Autonomous** ()

Autonomous should go here.

## virtual void **OperatorControl** ()

Operator control (tele-operated) code should go here.

virtual void **Test()**

Test program should go here.

## virtual void **RobotMain** ()

void **StartCompetition()**  
Robot main program for free-form programs.  
Start a competition.

---

# Detailed Description

## **Todo:**

If this is going to last until release, it needs a better name.

---

# Member Function Documentation

## **void SimpleRobot::Autonomous( ) [virtual]**

Autonomous should go here.

Programmers should override this method to run code that should run while the field is in the autonomous period. This will be called once each time the robot enters the autonomous state.

## **void SimpleRobot::Disabled( ) [virtual]**

Disabled should go here.

Programmers should override this method to run code that should run while the field is disabled.

## **void SimpleRobot::OperatorControl( ) [virtual]**

Operator control (tele-operated) code should go here.

Programmers should override this method to run code that should run while the field is in the Operator Control (tele-operated) period. This is called once each time the robot enters the teleop state.

## **void SimpleRobot::RobotInit( ) [virtual]**

Robot-wide initialization code should go here.

Programmers should override this method for default Robot-wide initialization which will be called each time the robot enters the disabled state.

## **void SimpleRobot::RobotMain( ) [virtual]**

Robot main program for free-form programs.

This should be overridden by user subclasses if the intent is to not use the **Autonomous()** and **OperatorControl()** methods. In that case, the program is responsible for sensing when to run the autonomous and operator control functions in

their program.

This method will be called immediately after the constructor is called. If it has not been overridden by a user subclass (i.e. the default version runs), then the **Autonomous()** and **OperatorControl()** methods will be called.

### **void SimpleRobot::StartCompetition( ) [virtual]**

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 or Test 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.

Implements **RobotBase**.

### **void SimpleRobot::Test( ) [virtual]**

Test program should go here.

Programmers should override this method to run code that executes while the robot is in test mode. This will be called once whenever the robot enters test mode

---

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

- C:/WindRiver/workspace/WPILib/**SimpleRobot.h**
- C:/WindRiver/workspace/WPILib/SimpleRobot.cpp

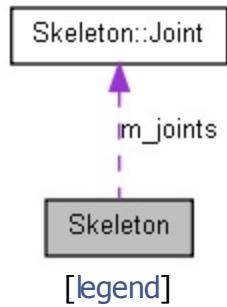


# Skeleton Class Reference

Represents **Skeleton** data from a **Kinect** device connected to the Driver Station.  
[More...](#)

```
#include <Skeleton.h>
```

Collaboration diagram for Skeleton:



[List of all members.](#)

# Classes

```
struct Joint
```

---

```
    JointTypes {
        HipCenter = 0, Spine = 1, ShoulderCenter = 2, Head = 3,
        ShoulderLeft = 4, ElbowLeft = 5, WristLeft = 6, HandLeft = 7,
        ShoulderRight = 8, ElbowRight = 9, WristRight = 10, HandRight =
enum 11,
        HipLeft = 12, KneeLeft = 13, AnkleLeft = 14, FootLeft = 15,
        HipRight = 16, KneeRight = 17, AnkleRight = 18, FootRight = 19,
        JointCount = 20
    }
enum JointTrackingState { kNotTracked, kInferred, kTracked }
```

# Public Member Functions

**Joint GetHandRight ()**  
**Joint GetHandLeft ()**  
**Joint GetWristRight ()**  
**Joint GetWristLeft ()**  
**Joint GetElbowLeft ()**  
**Joint GetElbowRight ()**  
**Joint GetShoulderLeft ()**  
**Joint GetShoulderRight ()**  
**Joint GetShoulderCenter ()**  
**Joint GetHead ()**  
**Joint GetSpine ()**  
**Joint GetHipCenter ()**  
**Joint GetHipRight ()**  
**Joint GetHipLeft ()**  
**Joint GetKneeLeft ()**  
**Joint GetKneeRight ()**  
**Joint GetAnkleLeft ()**  
**Joint GetAnkleRight ()**  
**Joint GetFootLeft ()**  
**Joint GetFootRight ()**  
**Joint GetJointValue (JointTypes index)**

# Friends

class **Kinect**

---

# Detailed Description

Represents **Skeleton** data from a **Kinect** device connected to the Driver Station.

See Getting Started with Microsoft **Kinect** for FRC and the **Kinect** for Windows SDK API reference for more information

---

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

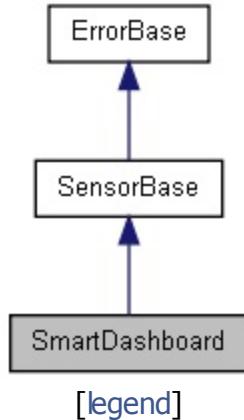
- C:/WindRiver/workspace/WPILib/**Skeleton.h**
- 

Generated by  1.7.2

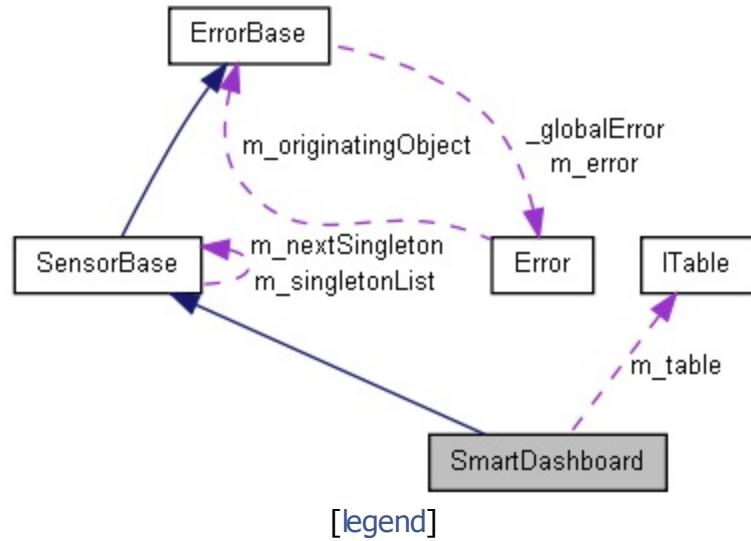


# SmartDashboard Class Reference

Inheritance diagram for SmartDashboard:



Collaboration diagram for SmartDashboard:



List of all members.

# Static Public Member Functions

static void	<b>init ()</b>
static void	<b>PutData (std::string key, Sendable *data)</b> Maps the specified key to the specified value in this table.
static void	<b>PutData (NamedSendable *value)</b> Maps the specified key (where the key is the name of the <b>SmartDashboardNamedData</b> ) to the specified value in this table.
static void	<b>PutBoolean (std::string keyName, bool value)</b> Maps the specified key to the specified value in this table.
static bool	<b>GetBoolean (std::string keyName)</b> Returns the value at the specified key.
static void	<b>PutNumber (std::string keyName, double value)</b> Maps the specified key to the specified value in this table.
static double	<b>GetNumber (std::string keyName)</b> Returns the value at the specified key.
static void	<b>PutString (std::string keyName, std::string value)</b> Maps the specified key to the specified value in this table.
static int	<b>GetString (std::string keyName, char *value, unsigned int valueLen)</b> Returns the value at the specified key.
static std::string	<b>GetString (std::string keyName)</b> Returns the value at the specified key.
static void	<b>PutValue (std::string keyName, ComplexData &amp;value)</b> Returns the value at the specified key.
static void	<b>RetrieveValue (std::string keyName, ComplexData &amp;value)</b> Retrieves the complex value (such as an array) in this table into the complex data object The key can not be NULL.

# Member Function Documentation

**bool SmartDashboard::GetBoolean ( std::string keyName ) [static]**

Returns the value at the specified key.

**Parameters:**

**keyName** the key

**Returns:**

the value

**double SmartDashboard::GetNumber ( std::string keyName ) [static]**

Returns the value at the specified key.

**Parameters:**

**keyName** the key

**Returns:**

the value

**int SmartDashboard::GetString ( std::string keyName,  
                                  char \* outBuffer,  
                                  unsigned int bufferLen  
                                 ) [static]**

Returns the value at the specified key.

**Parameters:**

**keyName** the key

**value** the buffer to fill with the value

**valueLen** the size of the buffer pointed to by value

**Returns:**

the length of the string

**std::string SmartDashboard::GetString ( std::string keyName ) [static]**

Returns the value at the specified key.

**Parameters:**

**keyName** the key

**Returns:**

the value

```
void SmartDashboard::PutBoolean ( std::string keyName,  
                                bool value [static]  
                                )
```

Maps the specified key to the specified value in this table.

The key can not be NULL. The value can be retrieved by calling the get method with a key that is equal to the original key.

**Parameters:**

**keyName** the key

**value** the value

```
void SmartDashboard::PutData ( std::string key,  
                             Sendable * data [static]  
                             )
```

Maps the specified key to the specified value in this table.

The key can not be NULL. The value can be retrieved by calling the get method with a key that is equal to the original key.

**Parameters:**

**keyName** the key

**value** the value

```
void SmartDashboard::PutData ( NamedSendable * value ) [static]
```

Maps the specified key (where the key is the name of the **SmartDashboardNamedData**) to the specified value in this table.

The value can be retrieved by calling the get method with a key that is equal to the original key.

#### Parameters:

**value** the value

```
void SmartDashboard::PutNumber ( std::string keyName,  
                                double      value  
                                )           [static]
```

Maps the specified key to the specified value in this table.

The key can not be NULL. The value can be retrieved by calling the get method with a key that is equal to the original key.

#### Parameters:

**keyName** the key  
**value** the value

```
void SmartDashboard::PutString ( std::string keyName,  
                                std::string value  
                                )           [static]
```

Maps the specified key to the specified value in this table.

Neither the key nor the value can be NULL. The value can be retrieved by calling the get method with a key that is equal to the original key.

#### Parameters:

**keyName** the key  
**value** the value

```
void SmartDashboard::PutValue ( std::string      keyName,  
                               ComplexData & value  
                               )           [static]
```

Returns the value at the specified key.

#### Parameters:

**keyName** the key

## Returns:

the value Maps the specified key to the specified complex value (such as an array) in this table. The key can not be NULL. The value can be retrieved by calling the RetrieveValue method with a key that is equal to the original key.

## Parameters:

**keyName** the key  
**value** the value

```
void SmartDashboard::RetrieveValue( std::string      keyName,  
                                  ComplexData & value  
                                )           [static]
```

Retrieves the complex value (such as an array) in this table into the complex data object The key can not be NULL.

## Parameters:

**keyName** the key  
**value** the object to retrieve the value into

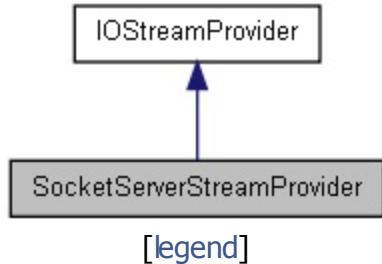
The documentation for this class was generated from the following files:

- C:/WindRiver/workspace/WPILib/SmartDashboard/**SmartDashboard.h**
- C:/WindRiver/workspace/WPILib/SmartDashboard/SmartDashboard.cpp

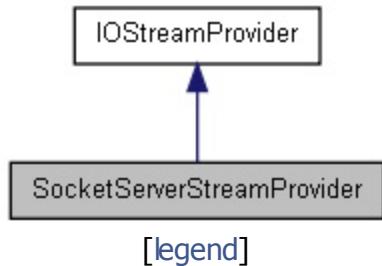


# SocketServerStreamProvider Class Reference

Inheritance diagram for SocketServerStreamProvider:



Collaboration diagram for SocketServerStreamProvider:



List of all members.

# Public Member Functions

**SocketServerStreamProvider** (int port)

**IOStream \*** **accept** ()

void **close** ()

Close the source of the IOStreams.

---

# Member Function Documentation

**IOStream \* SocketServerStreamProvider::accept( ) [virtual]**

## Returns:

a new **IOStream** normally from a server

## Exceptions:

**IOException**

Implements **IOStreamProvider**.

**void SocketServerStreamProvider::close( ) [virtual]**

Close the source of the IOStreams.

**accept()** should not be called after this is called

## Exceptions:

**IOException**

Implements **IOStreamProvider**.

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

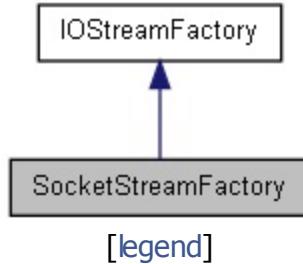
- C:/WindRiver/workspace/WPILib/networktables2/stream/**SocketServerStreamPro**
- C:/WindRiver/workspace/WPILib/networktables2/stream/SocketServerStreamProvide



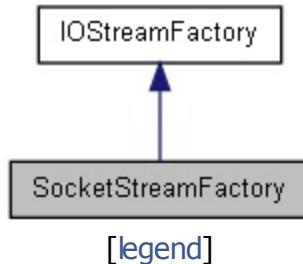
# SocketStreamFactory Class Reference

```
#include <SocketStreamFactory.h>
```

Inheritance diagram for SocketStreamFactory:



Collaboration diagram for SocketStreamFactory:



List of all members.

## Public Member Functions

**SocketStreamFactory** (const char \*host, int port)

**IOStream \*** **createStream** ()

---

# Detailed Description

**Author:**

mwills

---

# Member Function Documentation

**IOStream \* SocketStreamFactory::createStream( ) [virtual]**

## Returns:

create a new stream

## Exceptions:

**IOException**

Implements **IOStreamFactory**.

---

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

- C:/WindRiver/workspace/WPILib/networktables2/stream/**SocketStreamFactory.h**
- C:/WindRiver/workspace/WPILib/networktables2/stream/SocketStreamFactory.cpp



# SocketStreams Class Reference

---

Static factory for socket stream factories and providers. More...

```
#include <SocketStreams.h>
```

List of all members.

## Static Public Member Functions

static **IOStreamFactory** & **newStreamFactory** (const char \*host, int port)  
Create a new **IOStream** factory.

static **IOStreamProvider** & **newStreamProvider** (int port)  
Create a new **IOStream** provider.

---

# **Detailed Description**

Static factory for socket stream factories and providers.

## **Author:**

Mitchell

---

# Member Function Documentation

**IOStreamFactory & SocketStreams::newStreamFactory ( const char \* host,  
int port  
) [static]**

Create a new **IOStream** factory.

#### Parameters:

**host**  
**port**

#### Returns:

a **IOStreamFactory** that will create Socket Connections on the given host and port

#### Exceptions:

**IOException**

**IOStreamProvider & SocketStreams::newStreamProvider ( int port ) [static]**

Create a new **IOStream** provider.

#### Parameters:

**port**

#### Returns:

an **IOStreamProvider** for a socket server on the given port

#### Exceptions:

**IOException**

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

- C:/WindRiver/workspace/WPILib/networktables2/stream/**SocketStreams.h**
- C:/WindRiver/workspace/WPILib/networktables2/stream/SocketStreams.cpp

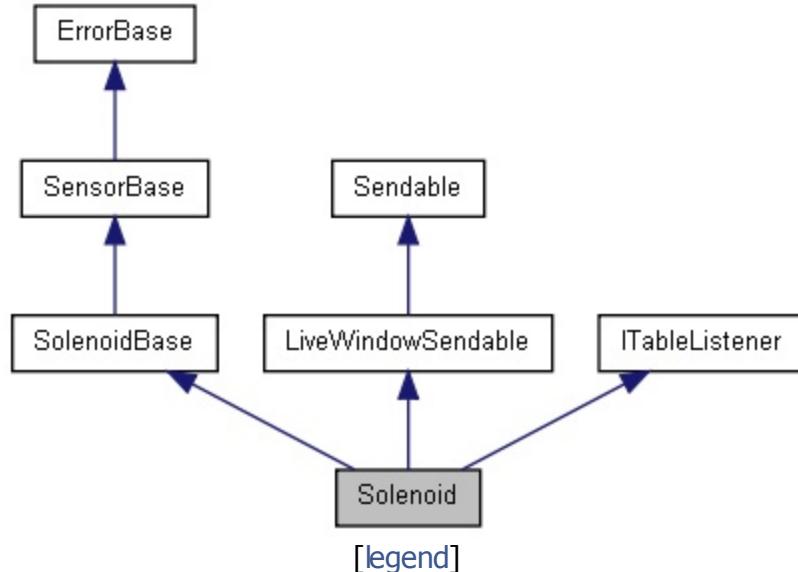


# Solenoid Class Reference

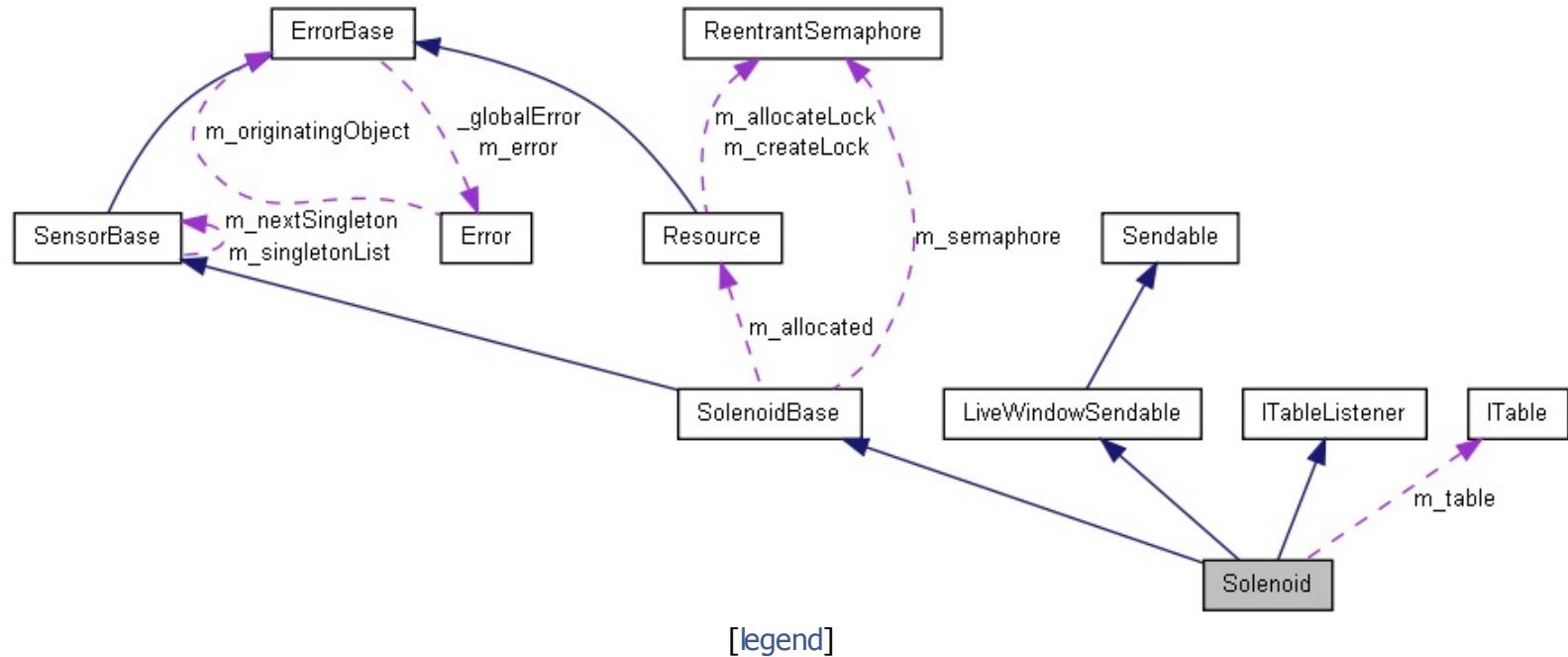
**Solenoid** class for running high voltage Digital Output (9472 module). [More...](#)

```
#include <Solenoid.h>
```

Inheritance diagram for Solenoid:



Collaboration diagram for Solenoid:



List of all members.

## Public Member Functions

**Solenoid** (UINT32 channel)

Constructor.

**Solenoid** (UINT8 moduleNumber, UINT32 channel)

Constructor.

**virtual ~Solenoid ()**

Destructor.

**virtual void Set (bool on)**

Set the value of a solenoid.

**virtual bool Get ()**

Read the current value of the solenoid.

**void ValueChanged (ITable \*source, const std::string &key, EntryValue value, bool isNew)**

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediately be called which could lead to recursive code.

**void UpdateTable ()**

Update the table for this sendable object with the latest values.

**void StartLiveWindowMode ()**

Start having this sendable object automatically respond to value changes reflect the value on the table.

**void StopLiveWindowMode ()**

Stop having this sendable object automatically respond to value changes.

**std::string GetSmartDashboardType ()**

**void InitTable (ITable \*subTable)**

Initializes a table for this sendable object.

**ITable \* GetTable ()**

## Detailed Description

**Solenoid** class for running high voltage Digital Output (9472 module).

The **Solenoid** class is typically used for pneumatics solenoids, but could be used for any device within the current spec of the 9472 module.

---

# Constructor & Destructor Documentation

## Solenoid::Solenoid ( **UINT32 channel** ) [explicit]

Constructor.

### Parameters:

**channel** The channel on the solenoid module to control (1..8).

## Solenoid::Solenoid ( **UINT8 moduleNumber,** **UINT32 channel** )

Constructor.

### Parameters:

**moduleNumber** The solenoid module (1 or 2).

**channel** The channel on the solenoid module to control (1..8).

# Member Function Documentation

## **bool Solenoid::Get( ) [virtual]**

Read the current value of the solenoid.

### **Returns:**

The current value of the solenoid.

## **std::string Solenoid::GetSmartDashboardType( ) [virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

## **ITable \* Solenoid::GetTable( ) [virtual]**

### **Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

## **void Solenoid::InitTable( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

## **void Solenoid::Set( bool on ) [virtual]**

Set the value of a solenoid.

### **Parameters:**

**on** Turn the solenoid output off or on.

```
void Solenoid::ValueChanged( ITable *  
                           const std::string & key,  
                           EntryValue value,  
                           bool isNew  
                           )  
                           source,  
                           [virtual]
```

Called when a key-value pair is changed in a **ITable** WARNING: If a new key-value is put in this method value changed will immediatly be called which could lead to recursive code.

#### Parameters:

**source** the table the key-value pair exists in

**key** the key associated with the value that changed

**value** the new value

**isNew** true if the key did not previously exist in the table, otherwise it is false

Implements **ITableListener**.

---

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

- C:/WindRiver/workspace/WPILib/**Solenoid.h**
- C:/WindRiver/workspace/WPILib/Solenoid.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

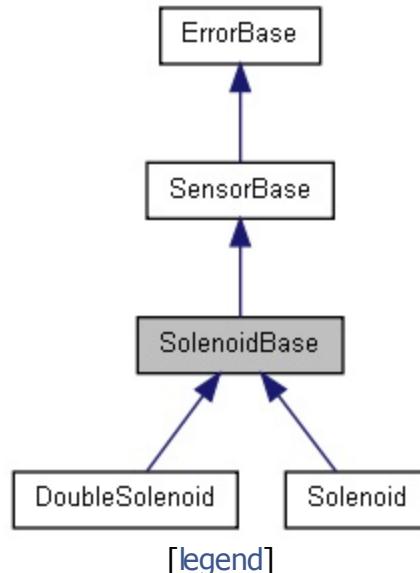
[Public Member Functions](#) |  
[Protected Member Functions](#) |  
[Protected Attributes](#) |  
[Static Protected Attributes](#)

# SolenoidBase Class Reference

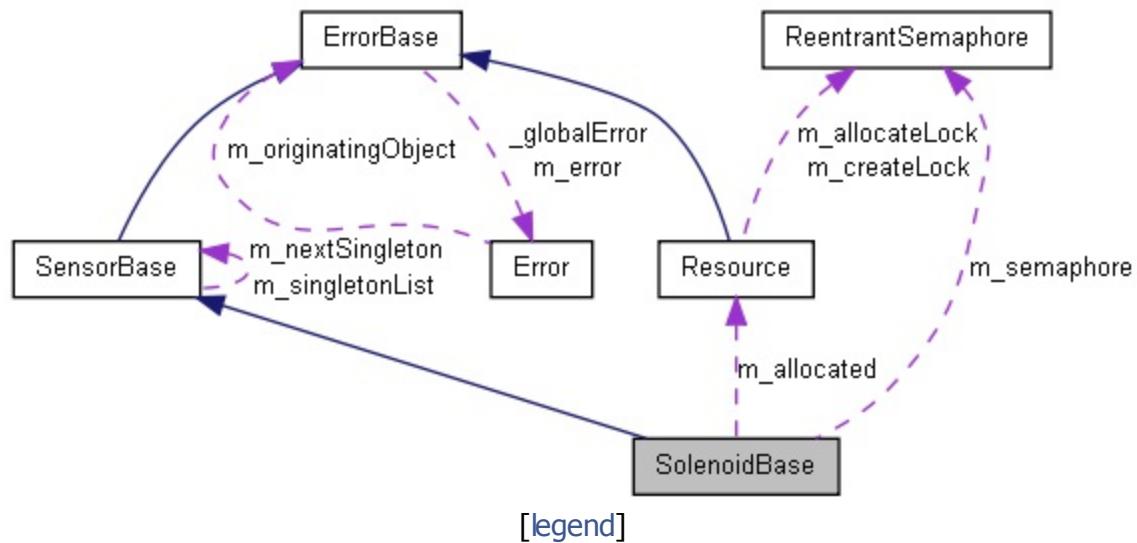
**SolenoidBase** class is the common base class for the **Solenoid** and **DoubleSolenoid** classes. [More...](#)

```
#include <SolenoidBase.h>
```

Inheritance diagram for SolenoidBase:



Collaboration diagram for SolenoidBase:



List of all members.

# Public Member Functions

virtual **[~SolenoidBase \(\)](#)**

Destructor.

UINT8 **[GetAll \(\)](#)**

Read all 8 solenoids as a single byte.

**[SolenoidBase \(UINT8 moduleNumber\)](#)**

Constructor.

void **[Set \(UINT8 value, UINT8 mask\)](#)**

Set the value of a solenoid.

virtual void **[InitSolenoid \(\)=0](#)**

## Protected Attributes

---

UINT32 **m\_moduleNumber**

Slot number where the module is plugged into the chassis.

## **Static Protected Attributes**

```
static Resource * m_allocated = NULL
```

---

## Detailed Description

**SolenoidBase** class is the common base class for the **Solenoid** and **DoubleSolenoid** classes.

---

# Constructor & Destructor Documentation

## SolenoidBase::SolenoidBase ( **UINT8 moduleNumber** ) [explicit, protected]

Constructor.

### Parameters:

**moduleNumber** The solenoid module (1 or 2).

## Member Function Documentation

## **UINT8 SolenoidBase::GetAll( )**

Read all 8 solenoids as a single byte.

## Returns:

The current value of all 8 solenoids on the module.

```
void SolenoidBase::Set( UINT8 value,  
                        UINT8 mask  
) [protected]
```

Set the value of a solenoid.

## Parameters:

**value** The value you want to set on the module.

**mask** The channels you want to be affected.

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

- C:/WindRiver/workspace/WPILib/**SolenoidBase.h**
  - C:/WindRiver/workspace/WPILib/SolenoidBase.cpp

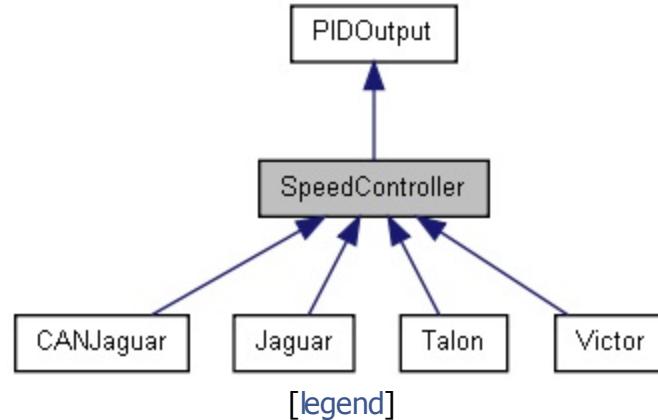


# SpeedController Class Reference

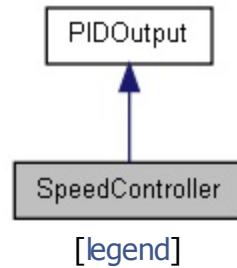
Interface for speed controlling devices. [More...](#)

```
#include <SpeedController.h>
```

Inheritance diagram for SpeedController:



Collaboration diagram for SpeedController:



[List of all members.](#)

## Public Member Functions

virtual void **Set** (float speed, UINT8 syncGroup=0)=0

Common interface for setting the speed of a speed controller.

virtual float **Get** ()=0

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

virtual void **Disable** ()=0

Common interface for disabling a motor.

---

## Detailed Description

Interface for speed controlling devices.

---

# Member Function Documentation

## **virtual float SpeedController::Get( ) [pure virtual]**

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

### Returns:

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

Implemented in **CANJaguar**, **Jaguar**, **Talon**, and **Victor**.

## **virtual void SpeedController::Set( float speed,                           UINT8 syncGroup = 0                           ) [pure virtual]**

Common interface for setting the speed of a speed controller.

### Parameters:

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

**syncGroup** The update group to add this **Set()** to, pending UpdateSyncGroup(). If 0, update immediately.

Implemented in **CANJaguar**, **Jaguar**, **Talon**, and **Victor**.

---

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

- C:/WindRiver/workspace/WPIlib/**SpeedController.h**

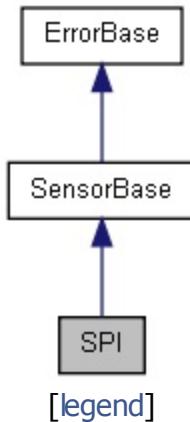
[Public Types](#) | [Public Member Functions](#) |  
[Protected Attributes](#) |  
[Static Protected Attributes](#)

# SPI Class Reference

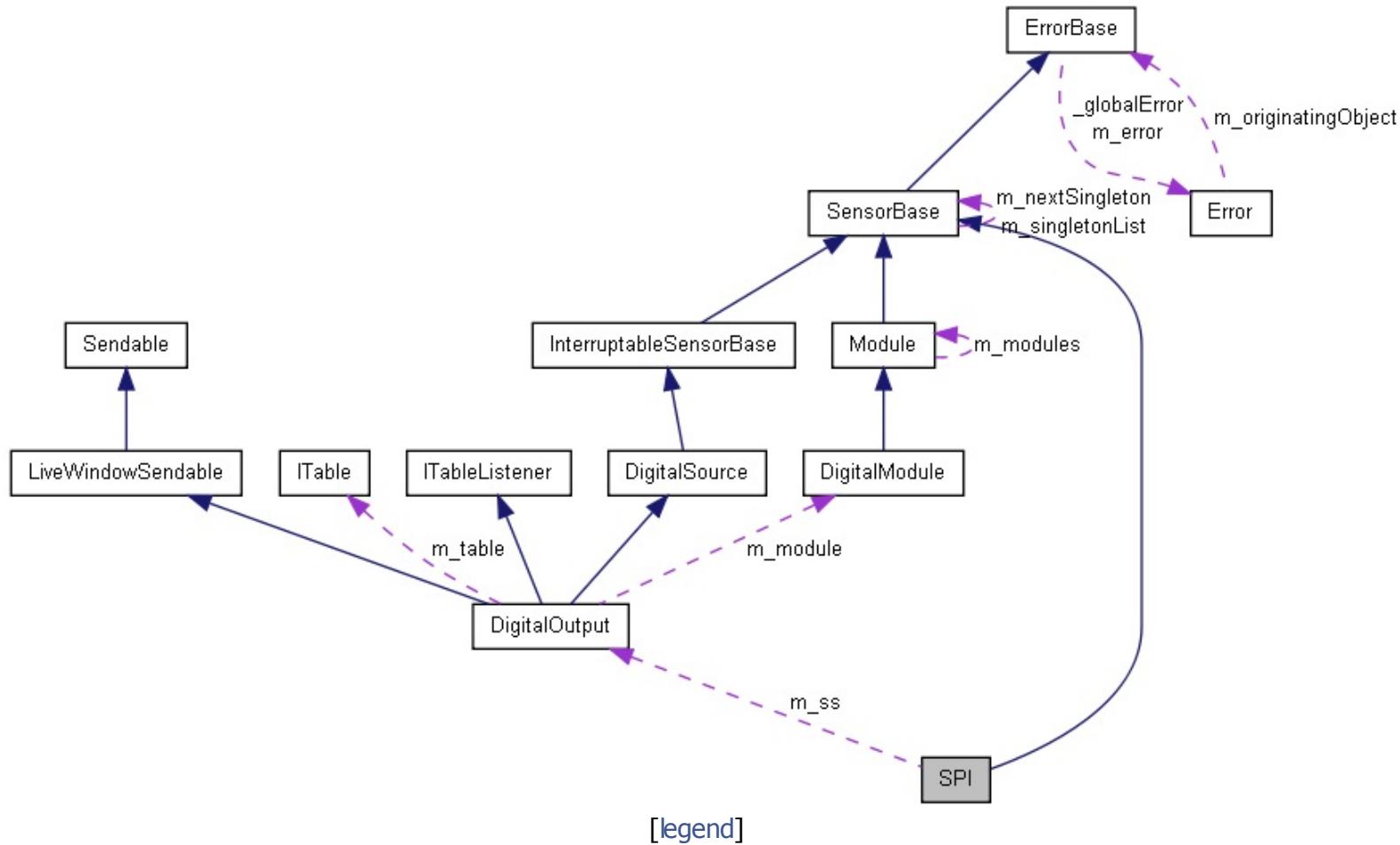
SPI bus interface class. More...

```
#include <SPI.h>
```

Inheritance diagram for SPI:



Collaboration diagram for SPI:



List of all members.

## Public Types

enum

**tFrameMode** { **kChipSelect**, **kPreLatchPulse**,  
**kPostLatchPulse**, **kPreAndPostLatchPulse** }

enum

**tSPIConstants** { **kReceiveFIFODepth** = 512,  
**kTransmitFIFODepth** = 512 }

## Public Member Functions

**SPI (DigitalOutput &clk, DigitalOutput &mosi, DigitalInput &miso)**

Constructor for input and output.

**SPI (DigitalOutput \*clk, DigitalOutput \*mosi, DigitalInput \*miso)**

Constructor for input and output.

**SPI (DigitalOutput &clk, DigitalOutput &mosi)**

Constructor for output only.

**SPI (DigitalOutput \*clk, DigitalOutput \*mosi)**

Constructor for output only.

**SPI (DigitalOutput &clk, DigitalInput &miso)**

Constructor for input only.

**SPI (DigitalOutput \*clk, DigitalInput \*miso)**

Constructor for input only.

virtual **~SPI ()**

Destructor.

void **SetBitsPerWord (UINT32 bits)**

Configure the number of bits from each word that the slave transmits or receives.

UINT32 **GetBitsPerWord ()**

Get the number of bits from each word that the slave transmits or receives.

void **SetClockRate (double hz)**

Configure the rate of the generated clock signal.

void **SetMSBFirst ()**

Configure the order that bits are sent and received on the wire to be most significant bit first.

void **SetLSBFirst ()**

Configure the order that bits are sent and received on the wire to be least significant bit first.

void **SetSampleDataOnFalling ()**

Configure that the data is stable on the falling edge and the data changes on the rising edge.

void **SetSampleDataOnRising ()**

Configure that the data is stable on the rising edge and the data changes on the falling edge.

void **SetSlaveSelect (DigitalOutput \*ss, tFrameMode mode=kChipSelect, bool activeLow=false)**

void	<b>SetSlaveSelect (DigitalOutput &amp;ss, tFrameMode mode=kChipSelect, bool activeLow=false)</b>	Configure the slave select line behavior.
<b>DigitalOutput *</b>	<b>GetSlaveSelect (tFrameMode *mode=NULL, bool *activeLow=NULL)</b>	Get the slave select line behavior.
void	<b>SetClockActiveLow ()</b>	Configure the clock output line to be active low.
void	<b>SetClockActiveHigh ()</b>	Configure the clock output line to be active high.
virtual void	<b>ApplyConfig ()</b>	Apply configuration settings and reset the <b>SPI</b> logic.
virtual UINT16	<b>GetOutputFIFOAvailable ()</b>	Get the number of words that can currently be stored before being transmitted to the device.
virtual UINT16	<b>GetNumReceived ()</b>	Get the number of words received and currently available to be read from the receive FIFO.
virtual bool	<b>IsDone ()</b>	Have all pending transfers completed?
bool	<b>HadReceiveOverflow ()</b>	Determine if the receive FIFO was full when attempting to add new data at end of a transfer.
virtual void	<b>Write (UINT32 data)</b>	Write a word to the slave device.
virtual UINT32	<b>Read (bool initiate=false)</b>	Read a word from the receive FIFO.
virtual void	<b>Reset ()</b>	Stop any transfer in progress and empty the transmit FIFO.
virtual void	<b>ClearReceivedData ()</b>	Empty the receive FIFO.

tSPI \* **m\_spi**

tSPI::tConfig **m\_config**

tSPI::tChannels **m\_channels**

**DigitalOutput \*** **m\_ss**

## Static Protected Attributes

```
static SEM_ID m_semaphore = NULL
```

# Detailed Description

**SPI** bus interface class.

This class is intended to be used by sensor (and other **SPI** device) drivers. It probably should not be used directly.

The FPGA only supports a single **SPI** interface.

---

# Constructor & Destructor Documentation

```
SPI::SPI ( DigitalOutput & clk,  
          DigitalOutput & mosi,  
          DigitalInput & miso  
        )
```

Constructor for input and output.

## Parameters:

- clk** The digital output for the clock signal.
- mosi** The digital output for the written data to the slave (master-out slave-in).
- miso** The digital input for the input data from the slave (master-in slave-out).

```
SPI::SPI ( DigitalOutput * clk,  
          DigitalOutput * mosi,  
          DigitalInput * miso  
        )
```

Constructor for input and output.

## Parameters:

- clk** The digital output for the clock signal.
- mosi** The digital output for the written data to the slave (master-out slave-in).
- miso** The digital input for the input data from the slave (master-in slave-out).

```
SPI::SPI ( DigitalOutput & clk,  
          DigitalOutput & mosi  
        )
```

Constructor for output only.

## Parameters:

- clk** The digital output for the clock signal.
- mosi** The digital output for the written data to the slave (master-out slave-in).

```
SPI::SPI ( DigitalOutput * clk,
```

```
DigitalOutput * mosi  
)
```

Constructor for output only.

**Parameters:**

- clk** The digital output for the clock signal.
- mosi** The digital output for the written data to the slave (master-out slave-in).

```
SPI::SPI ( DigitalOutput & clk,  
           DigitalInput & miso  
         )
```

Constructor for input only.

**Parameters:**

- clk** The digital output for the clock signal.
- miso** The digital input for the input data from the slave (master-in slave-out).

```
SPI::SPI ( DigitalOutput * clk,  
           DigitalInput * miso  
         )
```

Constructor for input only.

**Parameters:**

- clk** The digital output for the clock signal.
- miso** The digital input for the input data from the slave (master-in slave-out).

# Member Function Documentation

## **UINT32 SPI::GetBitsPerWord ( )**

Get the number of bits from each word that the slave transmits or receives.

### **Returns:**

The number of bits in one frame (1 to 32 bits).

## **UINT16 SPI::GetNumReceived ( ) [virtual]**

Get the number of words received and currently available to be read from the receive FIFO.

### **Returns:**

The number of words available to read.

## **UINT16 SPI::GetOutputFIFOAvailable ( ) [virtual]**

Get the number of words that can currently be stored before being transmitted to the device.

### **Returns:**

The number of words available to be written.

## **DigitalOutput \* SPI::GetSlaveSelect ( tFrameMode \* mode = NULL, bool \* activeLow = NULL )**

Get the slave select line behavior.

### **Parameters:**

**mode** Frame mode: kChipSelect: active for the duration of the frame.  
kPreLatchPulse: pulses before the transfer of each frame.  
kPostLatchPulse: pulses after the transfer of each frame.  
kPreAndPostLatchPulse: pulses before and after each frame.

**activeLow** True if slave select line is active low.

### **Returns:**

The slave select digital output.

## **bool SPI::HadReceiveOverflow( )**

Determine if the receive FIFO was full when attempting to add new data at end of a transfer.

### **Returns:**

True if the receive FIFO overflowed.

## **bool SPI::IsDone( ) [virtual]**

Have all pending transfers completed?

### **Returns:**

True if no transfers are pending.

## **UINT32 SPI::Read( bool initiate = false ) [virtual]**

Read a word from the receive FIFO.

Waits for the current transfer to complete if the receive FIFO is empty.

If the receive FIFO is empty, there is no active transfer, and initiate is false, errors.

### **Parameters:**

**initiate** If true, this function pushes "0" into the transmit buffer and initiates a transfer. If false, this function assumes that data is already in the receive FIFO from a previous write.

## **void SPI::SetBitsPerWord( UINT32 bits )**

Configure the number of bits from each word that the slave transmits or receives.

### **Parameters:**

**bits** The number of bits in one frame (1 to 32 bits).

## **void SPI::SetClockActiveHigh( )**

Configure the clock output line to be active high.

This is sometimes called clock polarity low.

### **void SPI::SetClockActiveLow( )**

Configure the clock output line to be active low.

This is sometimes called clock polarity high.

### **void SPI::SetClockRate( double hz )**

Configure the rate of the generated clock signal.

The default and maximum value is 76,628.4 Hz.

#### **Parameters:**

**hz** The clock rate in Hertz.

### **void SPI::SetSlaveSelect( DigitalOutput \* ss, tFrameMode mode = kChipSelect, bool activeLow = false )**

Configure the slave select line behavior.

#### **Parameters:**

**ss** slave select digital output.

**mode** Frame mode: kChipSelect: active for the duration of the frame.  
kPreLatchPulse: pulses before the transfer of each frame.  
kPostLatchPulse: pulses after the transfer of each frame.  
kPreAndPostLatchPulse: pulses before and after each frame.

**activeLow** True if slave select line is active low.

### **void SPI::SetSlaveSelect( DigitalOutput & ss, tFrameMode mode = kChipSelect, bool activeLow = false )**

Configure the slave select line behavior.

## Parameters:

<b>ss</b>	slave select digital output.
<b>mode</b>	Frame mode: kChipSelect: active for the duration of the frame. kPreLatchPulse: pulses before the transfer of each frame. kPostLatchPulse: pulses after the transfer of each frame. kPreAndPostLatchPulse: pulses before and after each frame.
<b>activeLow</b>	True if slave select line is active low.

## **void SPI::Write( **UINT32** data ) [virtual]**

Write a word to the slave device.

Blocks until there is space in the output FIFO.

If not running in output only mode, also saves the data received on the MISO input during the transfer into the receive FIFO.

---

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

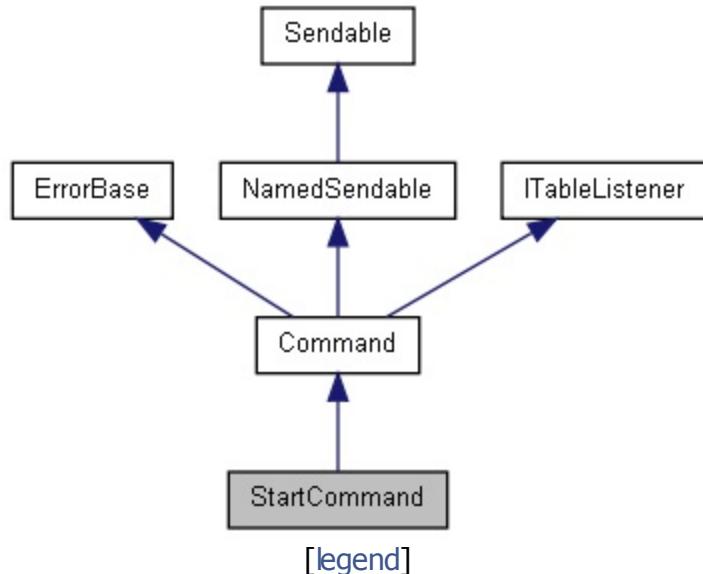
- C:/WindRiver/workspace/WPIlib/**SPI.h**
- C:/WindRiver/workspace/WPIlib/SPI.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

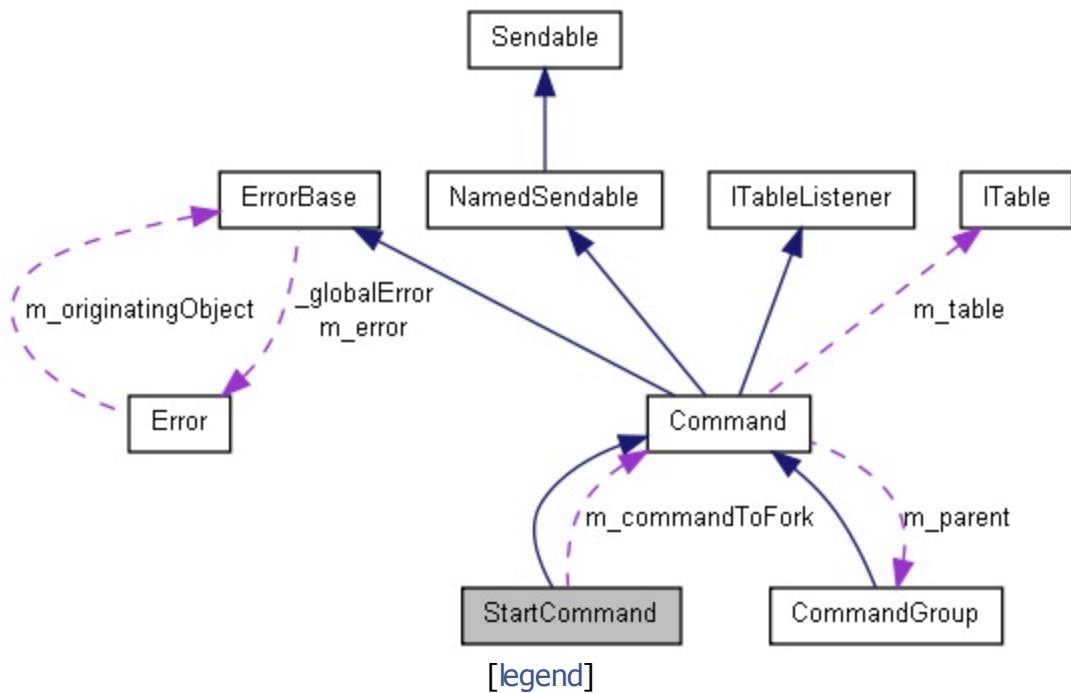
[Public Member Functions](#) |  
[Protected Member Functions](#)

# StartCommand Class Reference

Inheritance diagram for StartCommand:



Collaboration diagram for StartCommand:



List of all members.

# Public Member Functions

**StartCommand** (**Command** \*commandToStart)

**virtual void Initialize ()**

The initialize method is called the first time this **Command** is run after being started.

**virtual void Execute ()**

The execute method is called repeatedly until this **Command** either finishes or is canceled.

**virtual bool IsFinished ()**

Returns whether this command is finished.

**virtual void End ()**

Called when the command ended peacefully.

**virtual void Interrupted ()**

Called when the command ends because somebody called **cancel()** or another command shared the same requirements as this one, and booted it out.

# Member Function Documentation

## **void StartCommand::End( )** [protected, virtual]

Called when the command ended peacefully.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Implements [Command](#).

## **void StartCommand::Interrupted( )** [protected, virtual]

Called when the command ends because somebody called [cancel\(\)](#) or another command shared the same requirements as this one, and booted it out.

This is where you may want to wrap up loose ends, like shutting off a motor that was being used in the command.

Generally, it is useful to simply call the [end\(\)](#) method within this method

Implements [Command](#).

## **bool StartCommand::IsFinished( )** [protected, virtual]

Returns whether this command is finished.

If it is, then the command will be removed and [end\(\)](#) will be called.

It may be useful for a team to reference the [isTimedOut\(\)](#) method for time-sensitive commands.

### **Returns:**

whether this command is finished.

### **See also:**

[Command::isTimedOut\(\)](#) [isTimedOut\(\)](#)

Implements [Command](#).

- C:/WindRiver/workspace/WPILib/Commands/**StartCommand.h**
  - C:/WindRiver/workspace/WPILib/Commands/StartCommand.cpp
- 

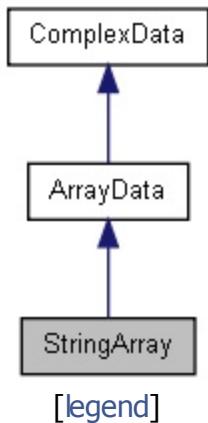
Generated by [doxygen](#) 1.7.2



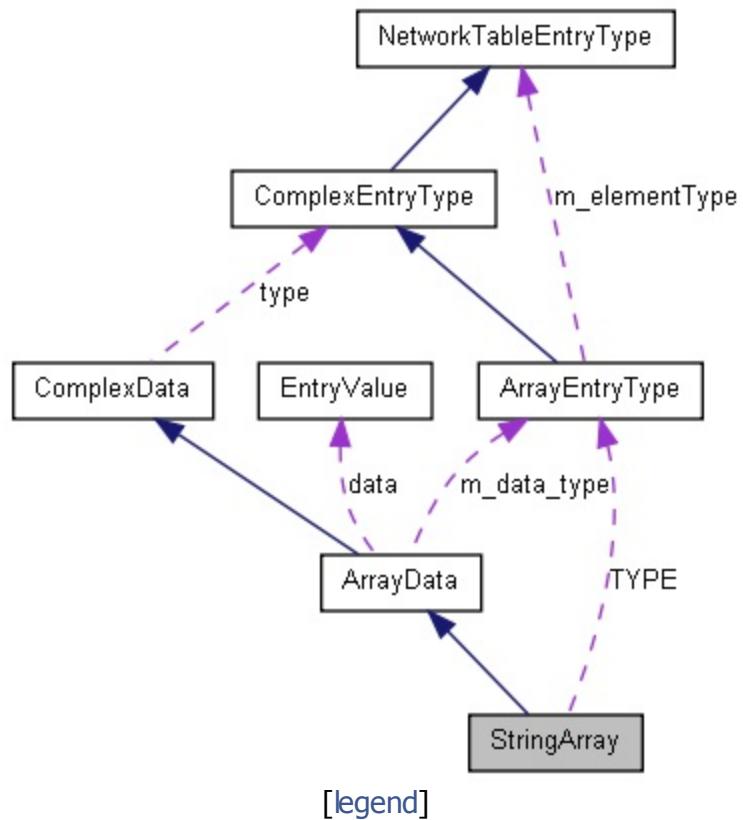
# StringArray Class Reference

```
#include <StringArray.h>
```

Inheritance diagram for StringArray:



Collaboration diagram for StringArray:



List of all members.

## Public Member Functions

```
std::string get (int index)
void set (int index, std::string value)
void add (std::string value)
```

```
static const TypeId STRING_ARRAY_RAW_ID = 0x12
static ArrayEntryType TYPE
```

---

# Detailed Description

## Author:

Mitchell

---

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

- C:/WindRiver/workspace/WPILib/networktables2/type/**StringArray.h**
- C:/WindRiver/workspace/WPILib/networktables2/type/StringArray.cpp

---

Generated by [doxygen](#) 1.7.2

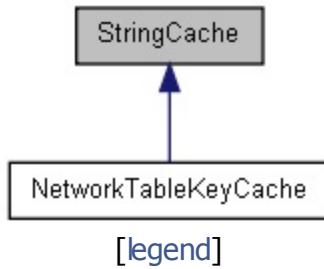


# StringCache Class Reference

A simple cache that allows for caching the mapping of one string to another calculated one. [More...](#)

```
#include <StringCache.h>
```

Inheritance diagram for StringCache:



[[legend](#)]

[List of all members.](#)

# Public Member Functions

## **StringCache ()**

std::string & **Get** (const std::string &input)

virtual std::string **Calc** (const std::string &input)=0

Will only be called if a value has not already been calculated.

---

## Detailed Description

A simple cache that allows for caching the mapping of one string to another calculated one.

### Author:

Mitchell

---

# Constructor & Destructor Documentation

## **StringCache::StringCache( )**

**Parameters:**

**input**

**Returns:**

the value for a given input

---

# Member Function Documentation

**virtual std::string StringCache::Calc ( const std::string & **input** ) [pure virtual]**

Will only be called if a value has not already been calculated.

## Parameters:

**input**

## Returns:

the calculated value for a given input

Implemented in **NetworkTableKeyCache**.

---

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

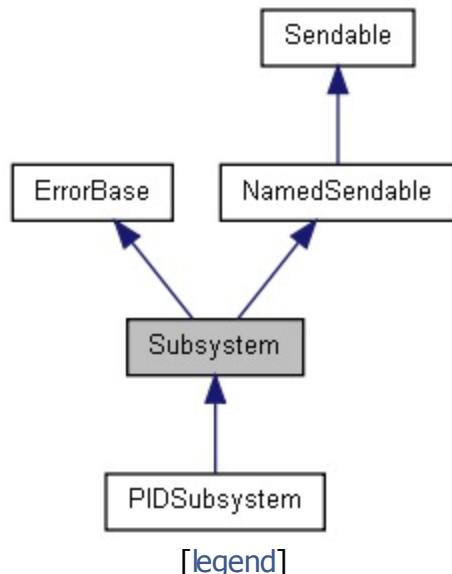
- C:/WindRiver/workspace/WPILib/networktables2/util/**StringCache.h**
- C:/WindRiver/workspace/WPILib/networktables2/util/StringCache.cpp

[Class List](#)[Class Hierarchy](#)[Class Members](#)

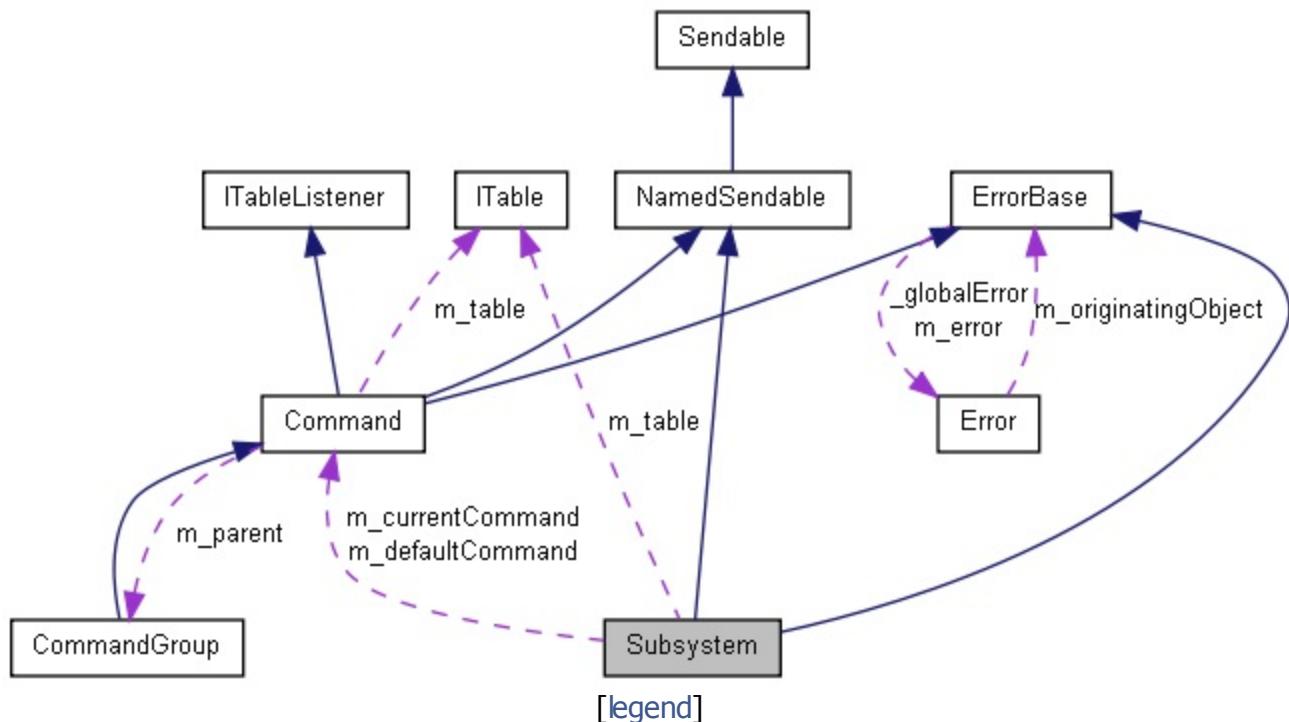
[Public Member Functions](#) |  
[Protected Attributes](#) | [Friends](#)

# Subsystem Class Reference

Inheritance diagram for Subsystem:



Collaboration diagram for Subsystem:



List of all members.

# Public Member Functions

**Subsystem** (const char \*name)

Creates a subsystem with the given name.

void **SetDefaultCommand** (**Command** \*command)

Sets the default command.

**Command** \* **GetDefaultCommand** ()

Returns the default command (or null if there is none).

void **SetCurrentCommand** (**Command** \*command)

Sets the current command.

**Command** \* **GetCurrentCommand** ()

Returns the command which currently claims this subsystem.

virtual void **InitDefaultCommand** ()

Initialize the default command for this subsystem. This is meant to be the place to call SetDefaultCommand in a subsystem and will be called on all the subsystems by the CommandBase method before the program starts running by using the list of all registered Subsystems inside the **Scheduler**.

virtual std::string **GetName** ()

virtual void **InitTable** (**ITable** \*table)

Initializes a table for this sendable object.

virtual **ITable** \* **GetTable** ()

virtual std::string **GetSmartDashboardType** ()

## Protected Attributes

**ITable \* m\_table**

class **Scheduler**

# Constructor & Destructor Documentation

## **Subsystem::Subsystem ( const char \* name )**

Creates a subsystem with the given name.

### **Parameters:**

**name** the name of the subsystem

# Member Function Documentation

## **Command \* Subsystem::GetCurrentCommand ( )**

Returns the command which currently claims this subsystem.

### **Returns:**

the command which currently claims this subsystem

## **Command \* Subsystem::GetDefaultCommand ( )**

Returns the default command (or null if there is none).

### **Returns:**

the default command

## **std::string Subsystem::GetName ( ) [virtual]**

### **Returns:**

the name of the subtable of **SmartDashboard** that the **Sendable** object will use

Implements **NamedSendable**.

## **std::string Subsystem::GetSmartDashboardType ( ) [virtual]**

### **Returns:**

the string representation of the named data type that will be used by the smart dashboard for this sendable

Implements **Sendable**.

Reimplemented in **PIDSubsystem**.

## **ITable \* Subsystem::GetTable ( ) [virtual]**

### **Returns:**

the table that is currently associated with the sendable

Implements **Sendable**.

## **void Subsystem::InitDefaultCommand ( ) [virtual]**

Initialize the default command for this subsystem. This is meant to be the place to call SetDefaultCommand in a subsystem and will be called on all the subsystems by the CommandBase method before the program starts running by using the list of all registered Subsystems inside the **Scheduler**.

This should be overridden by a **Subsystem** that has a default **Command**

## **void Subsystem::InitTable ( ITable \* subtable ) [virtual]**

Initializes a table for this sendable object.

### **Parameters:**

**subtable** The table to put the values in.

Implements **Sendable**.

Reimplemented in **PIDSubsystem**.

## **void Subsystem::SetCurrentCommand ( Command \* command )**

Sets the current command.

### **Parameters:**

**command** the new current command

## **void Subsystem::SetDefaultCommand ( Command \* command )**

Sets the default command.

If this is not called or is called with null, then there will be no default command for the subsystem.

**WARNING:** This should **NOT** be called in a constructor if the subsystem is a singleton.

### **Parameters:**

**command** the default command (or null if there should be none)

---

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

- C:/WindRiver/workspace/WPILib/Commands/**Subsystem.h**
- C:/WindRiver/workspace/WPILib/Commands/Subsystem.cpp

---

Generated by  1.7.2



# Synchronized Class Reference

---

Provide easy support for critical regions. [More...](#)

```
#include <Synchronized.h>
```

List of all members.

## Public Member Functions

**Synchronized** (SEM\_ID)

**Synchronized** class deals with critical regions.

**Synchronized** (**ReentrantSemaphore** &)

virtual **~Synchronized** ()

This destructor unlocks the semaphore.

---

## Detailed Description

Provide easy support for critical regions.

A critical region is an area of code that is always executed under mutual exclusion. Only one task can be executing this code at any time. The idea is that code that manipulates data that is shared between two or more tasks has to be prevented from executing at the same time otherwise a race condition is possible when both tasks try to update the data. Typically semaphores are used to ensure only single task access to the data.

**Synchronized** objects are a simple wrapper around semaphores to help ensure that semaphores are always unlocked (semGive) after locking (semTake).

You allocate a **Synchronized** as a local variable, \*not\* on the heap. That makes it a "stack object" whose destructor runs automatically when it goes out of scope. E.g.

```
{ Synchronized _sync(aReentrantSemaphore); ... critical region ... }
```

---

# Constructor & Destructor Documentation

## Synchronized::Synchronized ( SEM\_ID **semaphore** ) [explicit]

**Synchronized** class deals with critical regions.

Declare a **Synchronized** object at the beginning of a block. That will take the semaphore. When the code exits from the block it will call the destructor which will give the semaphore. This ensures that no matter how the block is exited, the semaphore will always be released. Use the CRITICAL\_REGION(SEM\_ID) and END\_REGION macros to make the code look cleaner (see header file)

### Parameters:

**semaphore** The semaphore controlling this critical region.

---

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

- C:/WindRiver/workspace/WPILib/**Synchronized.h**
- C:/WindRiver/workspace/WPILib/Synchronized.cpp



# TableKeyExistsWithDifferentTypeException Class Reference

---

Throw to indicate that an attempt to put data to a table is illegal because the specified key exists with a different data type than the put data type. [More...](#)

```
#include <TableKeyExistsWithDifferentTypeException.h>
```

[List of all members.](#)

## Public Member Functions

**TableKeyExistsWithDifferentTypeException** (const std::string existingKey, NetworkTableEntryType \*existingType)  
Creates a new **TableKeyExistsWithDifferentTypeException**.

**TableKeyExistsWithDifferentTypeException** (const std::string existingKey, NetworkTableEntryType \*existingType, const char \*message)

const char \* **what** ()

## Detailed Description

Throw to indicate that an attempt to put data to a table is illegal because the specified key exists with a different data type than the put data type.

### Author:

Paul Malmsten <[pmalmsten@gmail.com](mailto:pmalmsten@gmail.com)>

---

# Constructor & Destructor Documentation

## TableKeyExistsWithDifferentTypeException::TableKeyExistsWithDifferentTypeException

Creates a new **TableKeyExistsWithDifferentTypeException**.

### Parameters:

- existingKey** The name of the key which exists.
- existingType** The type of the key which exists.

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

- C:/WindRiver/workspace/WPILib/networktables2/**TableKeyExistsWithDifferentTypeException.h**
- C:/WindRiver/workspace/WPILib/networktables2/TableKeyExistsWithDifferentTypeEx