# SdsFalcon

Version 1.6.0

Sixdof Space

# Chapter 1

# Sixdof Falcon Documentation


## 1.1 Introduction

Sixdof Space has created an optical tracking solution offering tracking with cm level accuracy both indoors and outdoors in direct sunlight. Our patented technology leverages infrared lighting to serve as location beacons. The Falcon SDK contains algorithms that will report the 6dof positional data of the sensor relative to the beacons, 400 times a second. By connecting the sensor to your drone, robot, or VR headset, the Sixdof Falcon tracking system can provide full 6DOF tracking to your platform.

SdsFalcon is a shared library that enables users to interface with the Sixdof tracking technology. Drone guidance is the main application of this library, however other applications that require a programmatic interface to the Sixdof technology can use this library as well.

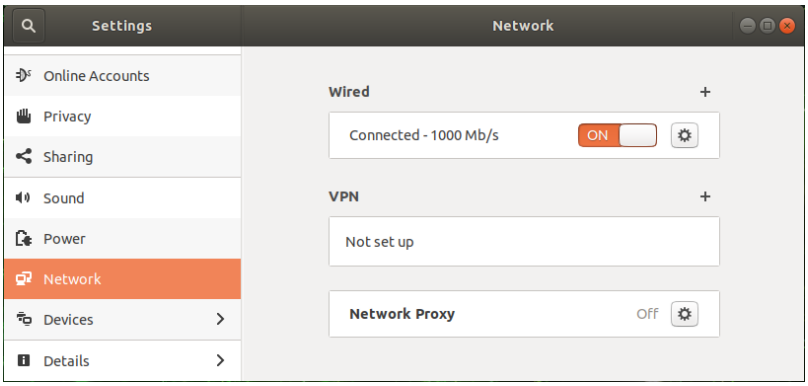
The SdsFalcon shared library is available on the following platforms:

- Windows
- Linux Ubuntu
- Linux Raspberry Pi OS (32 bit)
- Linux Raspberry Pi OS (64 bit)
- Nvidia Jetson

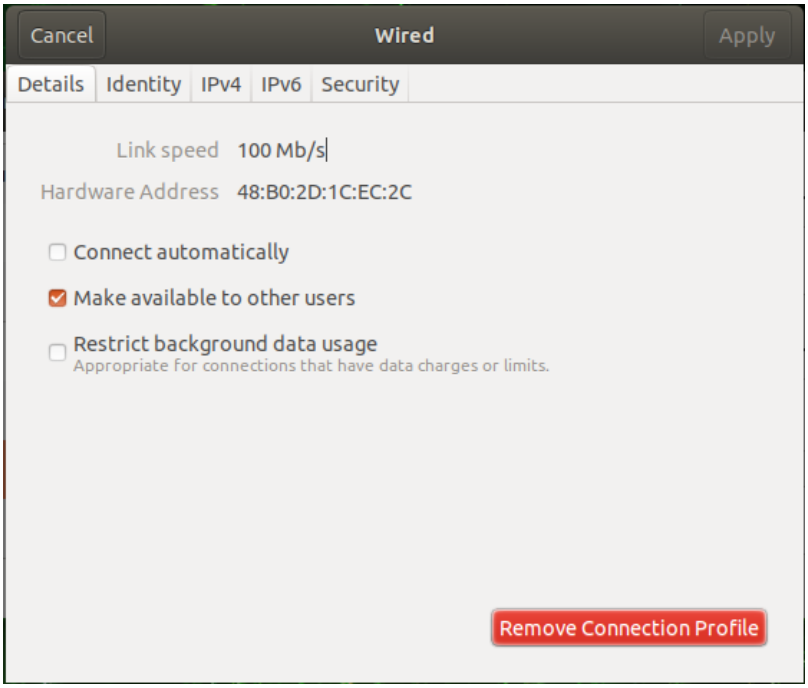
If you want to use the SdsFalcon on another platform, please contact the Sixdof team.

## 1.2 Network Setup

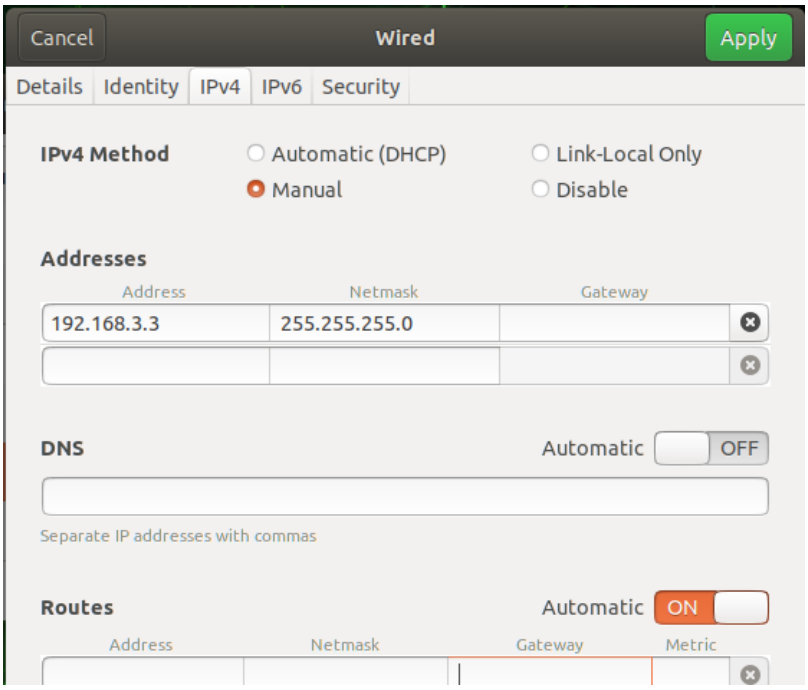
To connect the sensor to the host computer, first connect the sensor's micro-USB to a 5V power source. Next connect the RJ45 port to a network port on the host computer, or via a USB to Ethernet dongle. On Linux based computers, you must set up the network before operation of the sensor. Additionally, to achieve best performance on Linux platforms, Wifi should be turned off during operation. The net-tools package is required to connect to the sensor. You can test if you have the net-tools package by typing ¨ifconfig¨ into the command line, if the response says ¨command not found¨ please install the net-tools package via the command: ¨sudo apt install net-tools¨. The following instructions are for Ubuntu based distributions, the Raspberry Pi 3b instructions are shown below. First navigate to the Network Settings and ensure the ¨Wired¨ connection is set to ¨ON¨:



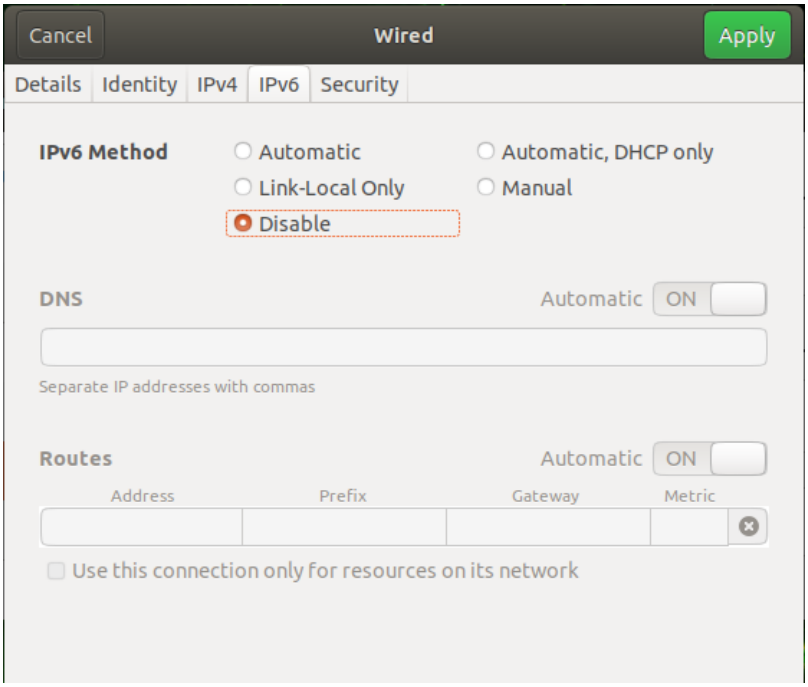

If you are using a USB dongle there will be another row labeled ¨USB Ethernet¨, set that to ¨ON¨. Next click the gear button to go to the advanced settings:



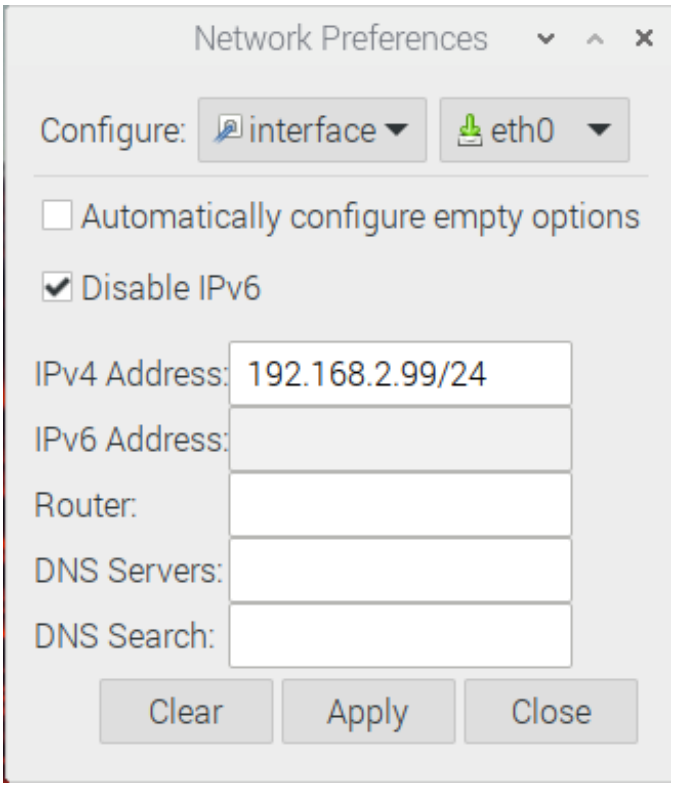

In the ¨Details¨ tab, deselect the checkbox that says ¨Connect automatically¨. Next navigate to the IPv4 tab and set the ¨IPv4 Method¨ to ¨Manual¨ and set the ¨Address¨ and ¨Netmask¨. The Address can be any IP address that you choose, however it should not clash with any other networks on the host computer. To avoid IP address clashes as much as possible you can set the Netmask to 255.255.255.0. Additionally, set the ¨DNS Automatic¨ to ¨OFF¨ as shown below:




Finally, navigate to the IPv6 tab and for the ¨IPv6 Method¨ select ¨Disable¨:




On Raspberry Pi 3b, navigate to the ¨Network Preferences¨ menu, next to ¨Configure¨ leave the first drop down on the ¨interface¨ option, and on the second drop down select the network that you are connected to, typically ¨eth0¨. If you are using a USB dongle the network may have a different name. Deselect the options that says ¨Automatically configure empty options¨, and select ¨Disable IPv6¨. Next fill in an IP Address that you selected and put ¨/24¨ at the end to indicate that you want a submask of 255.255.255.0. An example is shown below:




## 1.3 Coordinate Systems

Before using the Sixdof tracking system it is important to understand the coordinate systems involved. The sensor coordinate system is defined according to the image below:

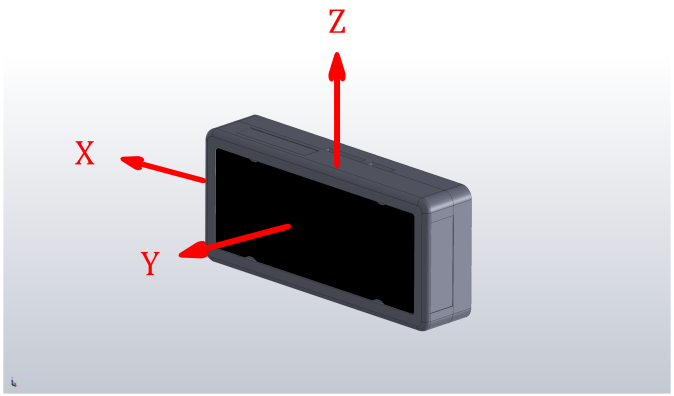



**Figure 1.1 Sensor coordinate system**

Where the positive x-axis points to the right of the sensor, the positive y-axis point out from the sensor and the positive z-axis completes the right handed system. The field of view can be approximated by a 60 degrees vision cone around the y-axis.

In many applications the user is interested in the location of the host platform (drone, or robot), and not just the location of the sensor. In these applications, it is the users responsibility to do the rigid body transformation from the sensor to the host platform, based on how the sensor is mounted on the platform.

## 1.4 Tracking Modes

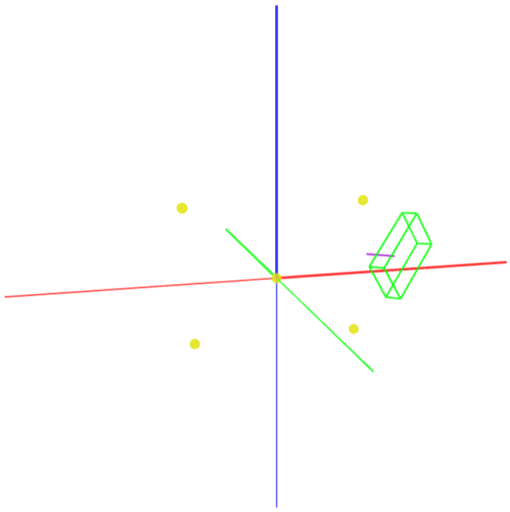
Three tracking modes are available:

- Sixdof: Track the full 6 degrees of freedom relative to the map.
- Relative Beacon: Track the position of a beacon relative to the sensor, this is intended for applications where the sensor is close to the beacons ($<$ 2m).
- Relative Angle: Track the angle of a beacon relative to the sensor, this is intended for applications where the sensor is far fromthe beacons ($>$ 2m) .

The SdsFalcon provides an easy way to run these two tracking modes, and seamlessly transition between them.

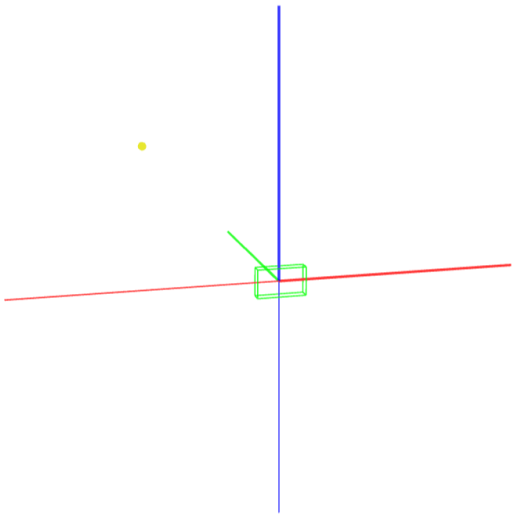
In addition to tracking, a method is provided to enable the user to do gyroscope calibration. During gyroscope calibration, the sensor must be still. In many applications, it is desirable to initialize the system while already in motion. Therefore, gyroscope calibration can be done in advance to the mission and the calibrated values can be fed in at run time. Alternatively, the gyro calibration step can be ignored, this is sufficient in most cases.

## 1.5 Tracking mode Sixdof



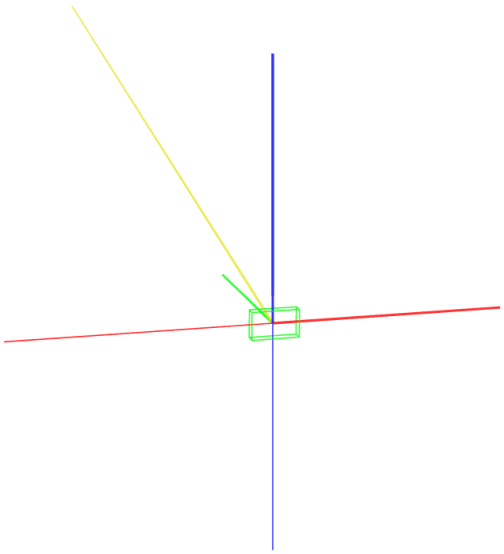
In this mode, the position of the sensor is tracked relative to the map. The map is made up of multiple IR beacons that flash unique ids, multiple beacons are driven by a single Basalt board. The user must place these beacons in a rigid formation and provide the X, Y, Z location of each beacon. The unique id of each beacon is provided on each LED individually. The user can define the coordinate system of the map in any way they choose, however it must be a right handed coordinate system. When designing a map, it is important to ensure the lights are not co-linear. Keep in mind the desired area of operation, try to design the map so that there will always be at least 3 beacons in the field of view of the sensor.

## 1.6 Tracking mode Relative Beacon



In this mode, the position of a beacon is tracked relative to the coordinate system of the sensor. However, this mode does not provide any orientation information. Due to the limited baseline of the sensor accurate tracking is only achievable in the near range ($\sim$2m). For mid and far range tracking of beacons see the Relative Angle mode below.

## 1.7 Tracking mode Relative Angle

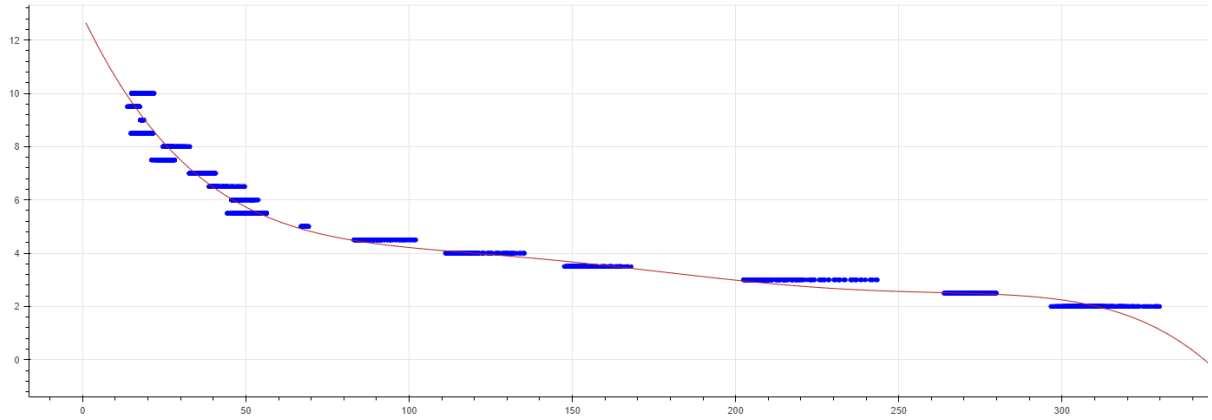


In this mode the angle of a beacon is tracked relative to the coordinate system of the sensor. For example, this mode can be used to guide a drone from a high elevation ($>$ 40m) all the way down to a landing pad. This mode does not provide any orientation information of the senor. Additionally, due to the far distance of the beacon the baseline

on the sensor is insufficient to calculate the distance directly. In this mode the intensity of the beacon and the peak width in pixels are reported in order to get a rough estimate of the distance of the beacon. In a configuration where full Sixdof landing is desired, the rough distance estimate can be used as a signal that the sensor is close enough to transition to Sixdof mode.

## 1.8 Switching between Tracking Modes

Switching between tracking modes can be done almost instantly via the setTrackingMode function. In many drone applications there is a need for the drone to be guided to the landing pad from a high elevation, as well as do a precision landing. In these cases a single high-powered LED can be used along side a map of multiple small LEDs. When the drone is at a high elevation it is guided to the landing target by using ¨Relative Angles¨ mode, then when it gets sufficiently close (∼6m) it switches to ¨Sixdof¨ mode for the final high-precision landing. The combination of ¨Relative Angles¨ and ¨Sixdof¨ modes is a highly effective way of achieving a targeted landing from a high elevation, however the key component to this solution is the ability to decided when to switch between the tracking modes. In the ¨Relative Angles¨ tracking modes the distance to the LED is unknown, however the intensity of the LED and the width in pixel space are provided. From this information a rough estimate of the distance to the landing target can be calculated. Shown below are the results of an experiment where the intensity of the LED was measured at known distances, y-axis is distance in meters and x-axis is the intensity, a polynomial trend line is shown in red:




A rough estimate of the distance between the sensor and the landing target can be obtained from the polynomial fit shown above. Additionally, the width in pixels of the LED can help to enhance the distance estimate when the sensor is close to the light. Note that this relationship is only valid for the beacon that was used in the experiment, in this case a 100 watt beacon, and the intensity to distance relationship can change slightly with lighting conditions. Many drones are equipped with other sensors such as range finders and barometers, these sensors can assist with estimating the distance to the landing target.

## 1.9 Change Log

| Version | Date | Notes |
|---|---|---|
| 1.6.0 | 21/07/2024 | Re-branded to Sixdof Falcon |
| 1.5.5 | 28/05/2024 | Added more elaborate check for the right firmware version, better network debugging tools, bug fix for Raspberry Pi platforms. |
| 1.5.3 | 29/02/2024 | Added NetworkAdvancedConfig, getFirmwareVersion, skip_gyro_calibration is now defaulted to true |
| 1.5.2 | 20/02/2024 | Added expected_beacon_ids to ConfigRelativeBeacon and ConfigRelativeAngle, also changed shutter backed, deprecated setShutter function |

| Version | Date | Notes |
|---|---|---|
| 1.5.1 | 01/02/2024 | Added skip_gyro_calibration flag to Config6Dof |
| 1.5.0 | 20/12/2023 | Added setShutter feature, current shutter callback, put shutter_settings into each of the config objects |
| 1.4.1 | 27/11/2023 | Added swapMap feature |
| 1.4.0 | 05/11/2023 | Added Relative Angle feature |
| 1.3.0 | 17/10/2023 | Added health monitoring feature in the form of Heartbeat callbacks |
| 1.2.1 | 06/09/2023 | Exposed ¨No Gyro¨ mode as a parameter in Config6Dof. Made changes to support multiple sensors each with its own FalconManager |
| 1.2.0 | 22/08/2023 | Changed API so that PoseRelativeBeacons that happen at the same time are bundled together |
| 1.1.2 | 14/08/2023 | Updated RelativeBeacon logic, parameter rel_lights_cooldown_ms is no longer needed, instead we have the parameter field_of_view_cutoff_deg |
| 1.1.1 | 14/08/2023 | Statically link GCC libraries, add MSVC build to the release |
| 1.1.0-beta | 24/07/2023 | Changed header, included gyro calibration, field of view report features and data logging for replay |
| 1.0.0-beta | 01/05/2023 | inital beta version |

## 1.10 Trouble Shooting Guide

### 1.10.1 Missing Libraries

Make sure to always copy the shared library (SdsFalcon.dll on Windows and SdsFalcon.so on Linux) that are provided in the release folder for your platform into the the directory with your program. Alternatively, you can statically link the shared library to your program. Also on Windows you have the option of adding the shared library to your system PATH.

### 1.10.2 Ping Failed

Errors relating to ping failing mean that the host computer was unable to connect to the Sixdof sensor. Try the following steps:

1. Ensure the Sixdof sensor is connected to power and the RJ45 ethernet cable is connected to your computer
2. Look at the red indicator lights that are exposed on the top of the senor casing, the lights should be flashing, if the lights are constant please contact the Sixdof team
3. Ensure your NetworkConfig parameters are correct, this is the sensor board number and the network name that the sensor is connected to
4. Your computers ARP table is used to connect to the sensor, try clearing the ARP table and trying again. The ARP table can be cleared vai a command console with administrator privileges, use the command ¨arp -d¨

### 1.10.3 Firewall

If SdsFalcon is reporting errors related to communication issues, it is recommended to turn off your firewall.

### 1.10.4 Underprocessing

If you are getting the ¨Warning¨ status message that says ¨Not receiving enough buffers from board¨ that means that your program is not getting enough CPU time. On Windows systems this is usually due to a spontaneous virus scan, you can temporarily turn off virus scans from in your ¨Windows Security¨ settings.

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 Sds::Falcon::Beacon Struct Reference

A structure representing a beacon with 3D coordinates and an id.

`#include <SdsFalcon.h>`

### Public Attributes

- double **x**

  *x-coordinate in meters.*
- double **y**

  *y-coordinate in meters.*
- double **z**

  *z-coordinate in meters.*
- uint16_t **id**

  *unique id.*

### 4.1.1 Detailed Description

A structure representing a beacon with 3D coordinates and an id.

This structure contains the 3D coordinates (x, y, z) of a beacon in meters and its unique id. Both the small beacons used for 6dof tracking, and the large beacon used for Relative Light tracking are represented with the Beacon struct. Each beacon provided is labeled with its unique id.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.2 Sds::Falcon::Config6Dof Struct Reference

A structure representing configuration settings for the Sixdof tracking mode.

`#include <SdsFalcon.h>`

**Public Attributes**

- std::vector< Beacon > **map**

  *vector of beacons for Sixdof tracking. (Required)*
- std::vector< GyroOffset > **gyroCalibration**

  *vector of previously recorded gyro calibration data. (Optional)*
- bool **skip_gyro_calibration** { true }

  *set to true to skip gyro calibration, this will speed up the initialization process.*
- ShutterSettings **shutter_settings** { getShutterSettingsAuto() }
- bool **no_gyro_mode** { false }

  *flag to disable gyro sensor, this mode should only be used in select cases, please contact the Sixdof team for guidance. (Optional)*
- std::string **sixdof_dump_path** { "" }

  *full path to file to log data to, default value of empty string will not log data. (Optional)*

### 4.2.1 Detailed Description

A structure representing configuration settings for the Sixdof tracking mode.

This structure contains configuration settings for the Sixdof tracking mode. The only required data is the map, other fields can be left to defaults.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.3 Sds::Falcon::ConfigGyroCalibration Struct Reference

A structure representing configuration settings for gyro calibration.

`#include <SdsFalcon.h>`

**Public Attributes**

- int **duration_seconds** { 30 }

  *duration of gyro calibration capture in seconds.*

### 4.3.1 Detailed Description

A structure representing configuration settings for gyro calibration.

This structure contains configuration settings for gyro calibration, all members can be left to defaults.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.4 Sds::Falcon::ConfigRelativeAngle Struct Reference

A structure representing configuration settings for RelativeAngle tracking mode.

`#include <SdsFalcon.h>`

### Public Attributes

- ShutterSettings **shutter_settings** { getShutterSettingsAuto() }
- std::vector< uint16_t > **expected_beacon_ids**

  *expected beacon ids for this mode, typically a large beacon. This information will be used to optimize the shutter for the beacons you are looking for. (Optional)*
- bool **matching_mode_single_beacon** { false }

  *set the matching mode to match based on the assumption that there is only one beacon in the feild of view. It is possible to increase the tracking range with this mode.*
- float **field_of_view_cutoff_deg** { 45.0 }

  *field of view cutoff for RelativeAngle mode, in degrees. (Optional)*
- std::string **rel_angles_dump_path** { "" }

  *full path to file to log data to, default value of empty string will not log data. (Optional)*

### 4.4.1 Detailed Description

A structure representing configuration settings for RelativeAngle tracking mode.

This structure contains configuration settings for RelativeAngle tracking mode. All members can be left to default values.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.5 Sds::Falcon::ConfigRelativeBeacon Struct Reference

A structure representing configuration settings for RelativeBeacon tracking mode.

`#include <SdsFalcon.h>`

### Public Attributes

- ShutterSettings **shutter_settings** { getShutterSettingsAuto() }
- std::vector< uint16_t > **expected_beacon_ids**

  *expected beacon ids for this mode, typically a large beacon. This information will be used to optimize the shutter for the beacons you are looking for. (Optional)*
- float **field_of_view_cutoff_deg** { 45.0 }

  *field of view cutoff for RelativeBeacon mode, in degrees. (Optional)*
- std::string **rel_lights_dump_path** { "" }

  *full path to file to log data to, default value of empty string will not log data. (Optional)*

### 4.5.1 Detailed Description

A structure representing configuration settings for RelativeBeacon tracking mode.

This structure contains configuration settings for RelativeBeacon tracking mode. All members can be left to default values.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.6 Sds::Falcon::CurrentShutter Struct Reference

A structure representing the current shutter value.

`#include <SdsFalcon.h>`

#### Public Attributes

- uint8_t **shutter_value**
  *current shutter value*

### 4.6.1 Detailed Description

A structure representing the current shutter value.

This structure is output via a callback, and indicates the current shutter value. Shutter values are between 0 and 16, where 0 is the shortest shutter and 16 is the longest.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.7 Sds::Falcon::FalconManager Class Reference

Class for managing the Sixdof Falcon tracking system.

`#include <SdsFalcon.h>`

### Public Member Functions

- virtual bool initializeNetwork (const NetworkConfig &)=0

  *Initialize the network.*
- virtual bool initializeNetwork (const NetworkAdvancedConfig &)=0

  *Initialize the network.*
- virtual bool initialize6Dof (const Config6Dof &)=0

  *Initialize Sixdof tracking mode.*
- virtual bool initializeRelativeBeacon (const ConfigRelativeBeacon &)=0

  *Initialize RelativeBeacon tracking mode.*
- virtual bool initializeRelativeAngle (const ConfigRelativeAngle &)=0

  *Initialize RelativeAngle tracking mode.*
- virtual void setTrackingMode (const TrackingMode &)=0

  *Set the tracking mode.*
- virtual Version **getFirmwareVersion** ()=0

  *Gets the firmware version of the sensor.*
- virtual std::vector< GyroOffset > getGyroCalibration (const ConfigGyroCalibration &)=0

  *Do gyro calibration and retrieve the results.*
- virtual Pose6Dof **getPose6Dof** ()=0

  *get most recent Pose6Dof.*
- virtual PoseRelativeBeaconCollection **getPoseRelativeBeaconCollection** ()=0

  *get most recent PoseRelativeBeaconCollection.*
- virtual RelativeAngleCollection **getRelativeAngleCollection** ()=0

  *get most recent RelativeAngle.*
- virtual StatusMessage **getStatusMessage** ()=0

  *get first StatusMessage that was called but not received yet.*
- virtual FieldOfViewReport **getFieldOfViewReport** ()=0

  *get most recent field of view report.*
- virtual CurrentShutter **getCurrentShutter** ()=0

  *get most recent shutter value.*
- virtual CallbackHandle **registerPose6DofCallback** (const std::function< void(Pose6Dof)> &)=0

  *register a function callback that takes Pose6Dof as an input.*
- virtual CallbackHandle **registerPoseRelativeBeaconCallback** (const std::function< void(PoseRelativeBeaconCollection)> &)=0

  *register a function callback that takes PoseRelativeBeaconCollection as an input.*
- virtual CallbackHandle **registerRelativeAngleCallback** (const std::function< void(RelativeAngleCollection)> &)=0

  *register a function callback that takes RelativeAngleCollection as an input.*
- virtual CallbackHandle **registerMessageCallback** (const std::function< void(StatusMessage)> &)=0

  *register a function callback that takes StatusMessage as an input.*
- virtual CallbackHandle **registerFieldOfViewReportCallback** (const std::function< void(FieldOfViewReport)> &)=0

  *register a function callback that takes FieldOfViewReport as an input.*
- virtual CallbackHandle **registerHeartbeatCallback** (const std::function< void(Heartbeat)> &)=0

  *register a function callback that takes Heartbeat as an input.*
- virtual CallbackHandle **registerCurrentShutterCallback** (const std::function< void(CurrentShutter)> &)=0

  *register a function callback that takes a CurrentShutter as an input.*
- virtual void **removePose6DofCallback** (const CallbackHandle &)=0

  *< remove a Pose6Dof function callback.*
- virtual void **removePoseRelativeBeaconCallback** (const CallbackHandle &)=0

  *< remove a PoseRelativeBeacon function callback.*

- virtual void **removeRelativeAngleCallback** (const CallbackHandle &)=0

  < *remove a RelativeAngle function callback.*
- virtual void **removeMessageCallback** (const CallbackHandle &)=0

  < *remove a StatusMessage function callback.*
- virtual void **removeFieldOfViewCallback** (const CallbackHandle &)=0

  < *remove a FieldOfViewReport function callback.*
- virtual void **removeHeartbeatCallback** (const CallbackHandle &)=0

  < *remove a Heartbeat function callback.*
- virtual void **removeCurrentShutterCallback** (const CallbackHandle &)=0

  < *remove a Current Shutter function callback.*
- virtual void swapMap (const std::vector< Beacon > &map)=0

  *Dynamically swap the map.*

### 4.7.1 Detailed Description

Class for managing the Sixdof Falcon tracking system.

This class provides an interface for managing Sixdof Falcon tracking. The correct order of function calls is first initializeNetwork, then initialize6dof, initializeRelativeBeacon, initializeRelativeAngle or multiple.

Finally you can start the tracking with the setTrackingMode function.

There are two ways to get tracking data and status messages out of the FalconManager:
1) Get functions. these will return the most recent data point. However the getStatusMessage function will return all messages in the order they appeared.
2) Function callbacks. The callbacks provided will get called each time a new data point is provided.

It is recommended to use function callbacks for status messages to ensure all messages are received immediately. It is also recommended to register for status messages before initializing the network, this way we can get all messages from the initialization stage. Each function callback is run on its own thread and it only gets updated once it has finished processing its current data point. This means that long running function callbacks may miss data points that got overshadowed by new data, this is by design.

The FalconManager also allows for the client to do gyro calibration before flight, see the getGyroCalibration() function.

### 4.7.2 Member Function Documentation

#### 4.7.2.1 getGyroCalibration()

```
virtual std::vector< GyroOffset > Sds::Falcon::FalconManager::getGyroCalibration (
            const ConfigGyroCalibration &  )  [pure virtual]
```

Do gyro calibration and retrieve the results.

This function retrieves gyro calibration data for the sensor. This data should be saved and used at a later time by passing it into the gyroCalibration member for Config6Dof

**Parameters**

| | |
|---|---|
| *ConfigGyroCalibration* | The configuration for the gyro calibration procedure. |

**Returns**

A vector of GyroOffset objects representing the gyro calibration data.

#### 4.7.2.2 initialize6Dof()

```
virtual bool Sds::Falcon::FalconManager::initialize6Dof (
            const Config6Dof &  )  [pure virtual]
```

Initialize Sixdof tracking mode.

This function initializes the Sixdof tracking mode. If you are not using the Sixdof tracking mode this is unnecessary.

**Parameters**

| | |
|---|---|
| *Config6Dof* | The configuration for the Sixdof tracking mode. |

**Returns**

True if Sixdof tracking is successfully initialized, false otherwise.

#### 4.7.2.3 initializeNetwork() [1/2]

```
virtual bool Sds::Falcon::FalconManager::initializeNetwork (
            const NetworkAdvancedConfig &  )  [pure virtual]
```

Initialize the network.

This function initializes the network configuration for Sixdof Falcon.

**Parameters**

| | |
|---|---|
| *NetworkAdvancedConfig* | The network configuration. |

**Returns**

True if the network is successfully initialized, false otherwise.

#### 4.7.2.4 initializeNetwork() [2/2]

```
virtual bool Sds::Falcon::FalconManager::initializeNetwork (
            const NetworkConfig &  )  [pure virtual]
```

Initialize the network.

This function initializes the network configuration for Sixdof Falcon.

**Parameters**

| | |
|---|---|
| *NetworkConfig* | The network configuration. |

**Returns**

True if the network is successfully initialized, false otherwise.

#### 4.7.2.5 initializeRelativeAngle()

```
virtual bool Sds::Falcon::FalconManager::initializeRelativeAngle (
            const ConfigRelativeAngle &  )  [pure virtual]
```

Initialize RelativeAngle tracking mode.

This function initializes the RelativeAngle tracking mode. If you are not using the RelativeAngle tracking mode this is unnecessary.

**Parameters**

| | |
|---|---|
| *ConfigRelativeAngle* | The configuration for RelativeAngle tracking mode. |

**Returns**

True if RelativeAngle tracking is successfully initialized, false otherwise.

#### 4.7.2.6 initializeRelativeBeacon()

```
virtual bool Sds::Falcon::FalconManager::initializeRelativeBeacon (
            const ConfigRelativeBeacon &  )  [pure virtual]
```

Initialize RelativeBeacon tracking mode.

This function initializes the RelativeBeacon tracking mode. If you are not using the RelativeBeacon tracking mode this is unnecessary.

**Parameters**

| | |
|---|---|
| *ConfigRelativeBeacon* | The configuration for RelativeBeacon tracking mode. |

**Returns**

True if RelativeBeacon tracking is successfully initialized, false otherwise.

#### 4.7.2.7 setTrackingMode()

```
virtual void Sds::Falcon::FalconManager::setTrackingMode (
            const TrackingMode &  )  [pure virtual]
```

Set the tracking mode.

**Parameters**

| | |
|---|---|
| *TrackingMode* | The desired TrackingMode. |

#### 4.7.2.8 swapMap()

```
virtual void Sds::Falcon::FalconManager::swapMap (
            const std::vector< Beacon > & map )  [pure virtual]
```

Dynamically swap the map.

This enables the user to swap the map during execution. It can only be called after initialize6Dof has already been called.

**Parameters**

| | |
|---|---|
| *map* | The new beacon positions. |

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.8 Sds::Falcon::FieldOfViewReport Struct Reference

A report of the beacons in the field of view.

```
#include <SdsFalcon.h>
```

### Public Attributes

- std::map< uint16_t, int > **seenBeacons**

  *Map of beacon ids to the number of optical sensors that the beacon was seen on.*

### 4.8.1 Detailed Description

A report of the beacons in the field of view.

This structure contains information about which beacons are in the field of view of the sensor, and if they are seen on 1, 2 or 3 optical sensors.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.9 Sds::Falcon::GyroOffset Struct Reference

A structure representing a single gyro calibration offset for a specific sensor.

`#include <SdsFalcon.h>`

### Public Attributes

- uint16_t **sensorId** { 0 }

  *sensor id.*
- std::string **datetime** { "" }

  *date time in Y-m-d H:M:S format.*
- int8_t **temperature** { 0 }

  *temperature in degrees. This is the internal temperature of board, not the temperature of the external environment.*
- int16_t **gyro_x** { 0 }

  *bias in the x-coordinate of the IMU. Units are in gyro units.*
- int16_t **gyro_y** { 0 }

  *bias in the y-coordinate of the IMU. Units are in gyro units.*
- int16_t **gyro_z** { 0 }

  *bias in the z-coordinate of the IMU. Units are in gyro units.*
- uint16_t **standard_dev** { 0 }

  *standard deviation of IMU noise. Units are in gyro units.*

### 4.9.1 Detailed Description

A structure representing a single gyro calibration offset for a specific sensor.

This structure contains information about gyro calibration offset for a specific sensor.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.10 Sds::Falcon::Heartbeat Struct Reference

A structure representing health data of the SDK.

`#include <SdsFalcon.h>`

### Public Attributes

- bool **sensor_communication_ok** { false }

  *status of communication with the sensor.*
- TrackingMode **current_tracking_mode** { TrackingMode::TrackingOFF }

  *current tracking mode.*
- TrackingModeState **current_tracking_mode_state** { TrackingModeState::Initializing }

  *state of the current tracking mode.*

### 4.10.1 Detailed Description

A structure representing health data of the SDK.

This structure is output via a callback every second. This enables the user to quickly identify that the SDK is still running. Additionally, the Heartbeat structure contains information regarding the state of communications with the sensor and the tracking algorithm.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.11 Sds::Falcon::NetworkAdvancedConfig Struct Reference

A structure representing advanced network configuration for the Sixdof sensor.

```
#include <SdsFalcon.h>
```

#### Public Attributes

- uint16_t **sensor_id**

  *sensor id.*
- std::string **sensor_ip**

  *desired sensor ip address.*
- uint16_t **udp_read_port** { 0 }

  *udp port to read from the sensor, to be auto-assigned a port leave as 0. (Optional)*

### 4.11.1 Detailed Description

A structure representing advanced network configuration for the Sixdof sensor.

This structure contains the necessary information in order to connect to a Sixdof sensor. This is different to the NetworkConfig object that only needs the sensor id and the network name. NetworkAdvancedConfig is for cases where the user will add the entry to the ARP table themselves. Typically this is used on system where the basic network configuration will fail, for example in situations where the computer operating system is not in English.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.12 Sds::Falcon::NetworkConfig Struct Reference

A structure representing network configuration for the Sixdof sensor.

```
#include <SdsFalcon.h>
```

**Public Attributes**

- uint16_t **sensor_id**

  *sensor id.*
- std::string **sensor_network_name**

  *name of the network the sensor is connected to.*
- uint16_t **udp_read_port** { 0 }

  *udp port to read from the sensor, to be auto-assigned a port leave as 0. (Optional)*

### 4.12.1 Detailed Description

A structure representing network configuration for the Sixdof sensor.

This structure contains the necessary information in order to connect to a Sixdof sensor.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.13 Sds::Falcon::Pose6Dof Struct Reference

A structure representing a 6dof pose.

`#include <SdsFalcon.h>`

**Public Attributes**

- bool **valid** { false }

  *flag indicating if the pose is valid. This can be False when calling the getPose6Dof function before a valid pose is obtained.*
- double **x**

  *x-coordinate in meters.*
- double **y**

  *y-coordinate in meters.*
- double **z**

  *z-coordinate in meters.*
- double **qw**

  *quaternion w component.*
- double **qx**

  *quaternion x component.*
- double **qy**

  *quaternion y component.*
- double **qz**

  *quaternion x component.*
- float **var_x**
- float **var_y**
- float **var_z**
- float **var_h**
- float **var_p**
- float **var_r**

### 4.13.1 Detailed Description

A structure representing a 6dof pose.

This structure contains 6dof (6 degree of freedom) pose information, specifically the pose of the Sixdof sensor with respect to the map. The pose is represented by position (x, y, z) in meters, and orientation as a quaternion (qx, qw, qy, qz). Pose accuracy is represented as variance of the position (var_x, var_y, var_z) in meters squared, and the variance of the orientation (var_h, var_p, var_r) in radians squared.

This is the return type when in Sixdof tracking mode.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.14 Sds::Falcon::PoseRelativeBeacon Struct Reference

A structure representing the relative pose of a beacon.

```
#include <SdsFalcon.h>
```

#### Public Attributes

- double **x**
  *x-coordinate in meters.*
- double **y**
  *y-coordinate in meters.*
- double **z**
  *z-coordinate in meters.*
- uint16_t **id**
  *unique id of the beacon.*

### 4.14.1 Detailed Description

A structure representing the relative pose of a beacon.

This structure contains the 3D pose of a single beacon with respect to the sensor. Pose is represented as 3D position (x, y, z) in meters.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.15 Sds::Falcon::RelativeAngle Struct Reference

A structure representing the relative angle of a beacon.

```
#include <SdsFalcon.h>
```

**Public Attributes**

- double **x_angle**

  *angular offset in the x-axis in radians.*
- double **z_angle**

  *angular offset in the z-axis in radians.*
- uint16_t **id**

  *unique id of the beacon.*
- double **intensity**

  *light intensity of the beacon, this can be used to roughly indicate distance.*
- double **width**

  *width of the detected peak in pixels, also useful for indicating distance.*

### 4.15.1 Detailed Description

A structure representing the relative angle of a beacon.

This structure contains the angle of a single beacon with respect to the sensor. Where x_angle is the angular offset along the x-axis, and z_angle is the angular offset along the z-axis.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.16 Sds::Falcon::ShutterSettings Struct Reference

Specifies the shutter settings.

`#include <SdsFalcon.h>`

**Public Attributes**

- uint8_t **min_shutter**
- uint8_t **max_shutter**

### 4.16.1 Detailed Description

Specifies the shutter settings.

This struct is used to control the sensors shutter. There are three shutter modes:

- Auto shutter - the shutter setting is detected automatically by the sensor.
- Fixed shutter - the shutter is fixed at a specific value and will not be changed by the sensor.
- Range shutter - the shutter is set automatically by the sensor but it will be fixed to a specific range.

For typical usage Auto shutter is recommended, Fixed and Range shutter modes are used only in cases where there are abnormal optical conditions. For example when the sun is directly in the field of view of the sensor. The ShutterSettings struct should be instantiated by one of the following functions: getShutterSettingsAuto, getShutter↩SettingsFixed, or getShutterSettingsRange.

Shutter values are between 0 and 16. Additionally the value 63 is used to set the sensor into Auto shutter mode.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.17 Sds::Falcon::StatusMessage Struct Reference

A structure representing a status message.

`#include <SdsFalcon.h>`

### Public Attributes

- Severity **severity**
  *severity level of the status message.*
- CallingLayer **layer**
  *calling layer for the status message.*
- uint8_t **event_code**
  *event code of the status message.*
- std::string **message**
  *verbose status message.*

### 4.17.1 Detailed Description

A structure representing a status message.

This structure contains information about a status message, including its severity, calling layer, event code, and the content of the message.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

## 4.18 Sds::Falcon::Version Class Reference

A class representing the version of either the shared library, or the sensor firmware.

`#include <SdsFalcon.h>`

### Public Member Functions

- **Version** (uint8_t major, uint8_t minor, uint8_t patch)
- uint8_t getMajor () const
  *Get the major component of the version.*
- uint8_t getMinor () const
  *Get the minor component of the version.*
- uint8_t getPatch () const
  *Get the patch component of the version.*
- bool isEqual (uint8_t major, uint8_t minor, uint8_t patch) const
  *Check if the version is equal to the specified components.*
- bool isAtLeast (uint8_t major, uint8_t minor, uint8_t patch) const
  *Check if the version is at least the specified version.*
- std::string getString () const
  *Get a string representation of the version.*

### 4.18.1 Detailed Description

A class representing the version of either the shared library, or the sensor firmware.

This class provides utility functions around a system version (major, minor, patch), either the version of the shared library, or the sensor firmware version.

### 4.18.2 Member Function Documentation

#### 4.18.2.1 getMajor()

```
uint8_t Sds::Falcon::Version::getMajor ( ) const
```

Get the major component of the version.

**Returns**

The major component of the version as a uint8_t.

#### 4.18.2.2 getMinor()

```
uint8_t Sds::Falcon::Version::getMinor ( ) const
```

Get the minor component of the version.

**Returns**

The minor component of the version as a uint8_t.

#### 4.18.2.3 getPatch()

```
uint8_t Sds::Falcon::Version::getPatch ( ) const
```

Get the patch component of the version.

**Returns**

The patch component of the version as a uint8_t.

#### 4.18.2.4 getString()

```
std::string Sds::Falcon::Version::getString ( ) const
```

Get a string representation of the version.

Get a string representation of the version.

**Returns**

A string representing the version.

#### 4.18.2.5 isAtLeast()

```
bool Sds::Falcon::Version::isAtLeast (
            uint8_t major,
            uint8_t minor,
            uint8_t patch ) const
```

Check if the version is at least the specified version.

**Parameters**

| | |
|---|---|
| *major* | The major component to compare. |
| *minor* | The minor component to compare. |
| *patch* | The patch component to compare. |

**Returns**

True if the version is at least the specified version, false otherwise.

#### 4.18.2.6 isEqual()

```
bool Sds::Falcon::Version::isEqual (
            uint8_t major,
            uint8_t minor,
            uint8_t patch ) const
```

Check if the version is equal to the specified components.

**Parameters**

| | |
|---|---|
| *major* | The major component to compare. |
| *minor* | The minor component to compare. |
| *patch* | The patch component to compare. |

**Returns**

True if the version is equal to the specified components, false otherwise.

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

- C:/sixdof/kiwi/src/falcon/SdsFalcon.h

# Chapter 5

# File Documentation

## 5.1 C:/sixdof/kiwi/src/falcon/SdsFalcon.h File Reference

Header file containing declarations for the Sixdof Falcon library.

```
#include <stdint.h>
#include <map>
#include <vector>
#include <string>
#include <memory>
#include <functional>
```

### Classes

- class Sds::Falcon::Version

  *A class representing the version of either the shared library, or the sensor firmware.*
- struct Sds::Falcon::NetworkConfig

  *A structure representing network configuration for the Sixdof sensor.*
- struct Sds::Falcon::NetworkAdvancedConfig

  *A structure representing advanced network configuration for the Sixdof sensor.*
- struct Sds::Falcon::Beacon

  *A structure representing a beacon with 3D coordinates and an id.*
- struct Sds::Falcon::Pose6Dof

  *A structure representing a 6dof pose.*
- struct Sds::Falcon::PoseRelativeBeacon

  *A structure representing the relative pose of a beacon.*
- struct Sds::Falcon::RelativeAngle

  *A structure representing the relative angle of a beacon.*
- struct Sds::Falcon::StatusMessage

  *A structure representing a status message.*
- struct Sds::Falcon::Heartbeat

  *A structure representing health data of the SDK.*
- struct Sds::Falcon::CurrentShutter

  *A structure representing the current shutter value.*
- struct Sds::Falcon::GyroOffset

*A structure representing a single gyro calibration offset for a specific sensor.*

- struct Sds::Falcon::FieldOfViewReport

  *A report of the beacons in the field of view.*
- struct Sds::Falcon::ShutterSettings

  *Specifies the shutter settings.*
- struct Sds::Falcon::Config6Dof

  *A structure representing configuration settings for the Sixdof tracking mode.*
- struct Sds::Falcon::ConfigRelativeBeacon

  *A structure representing configuration settings for RelativeBeacon tracking mode.*
- struct Sds::Falcon::ConfigRelativeAngle

  *A structure representing configuration settings for RelativeAngle tracking mode.*
- struct Sds::Falcon::ConfigGyroCalibration

  *A structure representing configuration settings for gyro calibration.*
- class Sds::Falcon::FalconManager

  *Class for managing the Sixdof Falcon tracking system.*

### Macros

- #define **SDS_API** __attribute__((visibility("default")))
- #define **DEPRECATED**(msg)

### Typedefs

- typedef std::vector< PoseRelativeBeacon > Sds::Falcon::PoseRelativeBeaconCollection

  *A structure representing a collection of PoseRelativeBeacon.*
- typedef std::vector< RelativeAngle > Sds::Falcon::RelativeAngleCollection

  *A structure representing a collection of RelativeAngle.*
- typedef uint32_t Sds::Falcon::CallbackHandle

  *A handle representing a registered callback function.*

### Enumerations

- enum class Sds::Falcon::Severity { Exception = 1 , Warning , Informative }

  *An enum representing the severity level of a status message.*
- enum class Sds::Falcon::CallingLayer {
  SdsFalcon = 1 , Algorithm6Dof , AlgorithmRelativeBeacon , SdsCommLib ,
  NetworkConnection , GyroCalibration , AlgorithmRelativeAngle }

  *An enum representing the calling layer for a status message.*
- enum class Sds::Falcon::FalconEventCode {
  UnexpectedAlgoInstance , TrackingModeNotInitalized , NetworkNotInitalized , GyroCalibrationRejected ,
  CommunicationLoopNotClosed , DidNotReceiveFirmwareInfo , SensorFirmwareIsNotLatest , SensorFirmwareIsNotCompatible
  }

  *An enum representing event codes specific to the SdsFalcon calling layer.*
- enum class Sds::Falcon::TrackingMode { TrackingOFF , RelativeBeacon , Sixdof , RelativeAngle }

  *An enum representing different tracking modes.*
- enum class Sds::Falcon::TrackingModeState { Initializing , Running , Error }

  *An enum representing different states of the current tracking mode.*

**Functions**

- SDS_API Version Sds::Falcon::getVersion ()

  *Get the version of the SdsFalcon shared library.*
- SDS_API std::string Sds::Falcon::to_string (TrackingMode mode)

  *Convert a TrackingMode enum value to a string.*
- SDS_API std::string Sds::Falcon::to_string (TrackingModeState state)

  *Convert a TrackingModeState enum value to a string.*
- SDS_API ShutterSettings Sds::Falcon::getShutterSettingsAuto ()

  *Get shutter settings for Auto shutter mode.*
- SDS_API ShutterSettings Sds::Falcon::getShutterSettingsFixed (uint8_t shutter)

  *Get shutter settings for Fixed shutter mode.*
- SDS_API ShutterSettings Sds::Falcon::getShutterSettingsRange (uint8_t min_shutter, uint8_t max_shutter)

  *Get shutter settings for Range shutter mode.*
- SDS_API std::vector< Beacon > Sds::Falcon::loadMap (const std::string &path)

  *Load a map file.*
- SDS_API std::unique_ptr< FalconManager > Sds::Falcon::createFalconManager ()

  *Create an instance of FalconManager.*

### 5.1.1 Detailed Description

Header file containing declarations for the Sixdof Falcon library.

This file contains the declarations for classes and data structures related to the Sixdof Falcon library. It provides functionality for managing the Sixdof sensor and the Sixdof Falcon tracking system.

### 5.1.2 Typedef Documentation

#### 5.1.2.1 CallbackHandle

`Sds::Falcon::CallbackHandle`

A handle representing a registered callback function.

This typedef represents a callback handle used to identify registered callback functions. It is used to remove a specific callback.

#### 5.1.2.2 PoseRelativeBeaconCollection

`Sds::Falcon::PoseRelativeBeaconCollection`

A structure representing a collection of PoseRelativeBeacon.

This structure contains multiple PoseRelativeBeacon estimates. All PoseRelativeBeacon estimates were calculated at the same and therefore bundled together.

This is the return type when in RelativeBeacon tracking mode.

#### 5.1.2.3 RelativeAngleCollection

`Sds::Falcon::RelativeAngleCollection`

A structure representing a collection of RelativeAngle.

This structure contains multiple RelativeAngle estimates. All RelativeAngle estimates were calculated at the same and therefore bundled together.

This is the return type when in RelativeAngle tracking mode.

### 5.1.3 Enumeration Type Documentation

#### 5.1.3.1 CallingLayer

`enum class Sds::Falcon::CallingLayer [strong]`

An enum representing the calling layer for a status message.

This enum defines the calling layers for status messages. Each layer indicates the source of a particular message.

**Enumerator**

| | |
|---|---|
| SdsFalcon | Message from the top layer of the library. Generally indicates the library was used incorrectly. |
| Algorithm6Dof | Message from Sixdof tracking mode. |
| AlgorithmRelativeBeacon | Message from RelativeBeacon tracking mode. |
| SdsCommLib | Message from the UDP communication library, this is used to communicate with the sensor. |
| NetworkConnection | Message from the network connection stage. |
| GyroCalibration | Message from the gyro calibration feature. |
| AlgorithmRelativeAngle | Message from RelativeAngle tracking mode. |

#### 5.1.3.2 FalconEventCode

`enum class Sds::Falcon::FalconEventCode [strong]`

An enum representing event codes specific to the SdsFalcon calling layer.

This enum defines event codes that are specific to the SdsFalcon calling layer.

**Enumerator**

| | |
|---|---|
| UnexpectedAlgoInstance | indicates internal error, contact Sixdof for tech support. |
| TrackingModeNotInitalized | indicates tracking mode was set before the corresponding initialization. |
| NetworkNotInitalized | indicates that the initializeNetwork function was not called. |
| GyroCalibrationRejected | indicates gyro calibration values were rejected. |
| CommunicationLoopNotClosed | communication loop is not closed, check your firewall. |
| DidNotReceiveFirmwareInfo | did not receive firmware information from the sensor. |

**Enumerator**

| | |
|---|---|
| SensorFirmwareIsNotLatest | sensor firmware is not latest version, it is not guaranteed that all SDK features will work fully. |
| SensorFirmwareIsNotCompatible | sensor firmware is not compatible with current SDK, please contact Sixdof for an upgrade. |

#### 5.1.3.3 Severity

```
enum class Sds::Falcon::Severity  [strong]
```

An enum representing the severity level of a status message.

This enumeration defines severity levels for status messages.

**Enumerator**

| | |
|---|---|
| Exception | Exception indicates that there was a error in operation. |
| Warning | Warning indicates that something went wrong but it is not critical. |
| Informative | Informative indicates general debugging information, these can be ignored. |

#### 5.1.3.4 TrackingMode

```
enum class Sds::Falcon::TrackingMode  [strong]
```

An enum representing different tracking modes.

This enum is used to set the tracking mode.

**Enumerator**

| | |
|---|---|
| TrackingOFF | tracking is turned off. This is the default and when the FalconManager goes out of scope it is set to TrackingOFF automatically. |
| RelativeBeacon | tracking is set to RelativeBeacon mode. |
| Sixdof | tracking is set to Sixdof (6dof) mode. |
| RelativeAngle | tracking is set to RelativeAngle mode. |

#### 5.1.3.5 TrackingModeState

```
enum class Sds::Falcon::TrackingModeState  [strong]
```

An enum representing different states of the current tracking mode.

**Enumerator**

| | |
|---|---|
| Initializing | tracking mode is currently initializing. |
| Running | tracking mode is currently running. |
| Error | tracking mode is currently in the error state. |

## 5.1.4 Function Documentation

### 5.1.4.1 createFalconManager()

```
SDS_API std::unique_ptr< FalconManager > Sds::Falcon::createFalconManager ( )
```

Create an instance of FalconManager.

This function creates a new instance of the FalconManager. When the FalconManager goes out of scope it will stop the sensor and be cleaned up automatically.

**Returns**

A unique pointer to a new instance of FalconManager.

### 5.1.4.2 getShutterSettingsAuto()

```
SDS_API ShutterSettings Sds::Falcon::getShutterSettingsAuto ( )
```

Get shutter settings for Auto shutter mode.

**Returns**

ShutterSettings

### 5.1.4.3 getShutterSettingsFixed()

```
SDS_API ShutterSettings Sds::Falcon::getShutterSettingsFixed (
            uint8_t shutter )
```

Get shutter settings for Fixed shutter mode.

**Parameters**

| | |
|---|---|
| *shutter* | Shutter value, between 0 and 16 inclusive. |

**Returns**

ShutterSettings

### 5.1.4.4 getShutterSettingsRange()

```
SDS_API ShutterSettings Sds::Falcon::getShutterSettingsRange (
            uint8_t min_shutter,
            uint8_t max_shutter )
```

Get shutter settings for Range shutter mode.

**Parameters**

| | |
|---|---|
| *min_shutter* | Min shutter value, between 0 and 16 inclusive. |
| *max_shutter* | Max shutter value, between 0 and 16 inclusive. |

**Returns**

ShutterSettings

#### 5.1.4.5 getVersion()

```
SDS_API Version Sds::Falcon::getVersion ( )
```

Get the version of the SdsFalcon shared library.

**Returns**

The Version object representing the library version.

#### 5.1.4.6 loadMap()

```
SDS_API std::vector< Beacon > Sds::Falcon::loadMap (
            const std::string & path )
```

Load a map file.

Load a CSV file with the map settings for the configuration of 6DOF tracking mode. This enables the user to load a file containing the map configuration instead of entering it manually. It can only be called before initialize6Dof is called. An exception is thrown if there is an error in loading the file.

**Parameters**

| | |
|---|---|
| *path* | The path to the file containing the map configuration. |

**Returns**

The map as a vector of Beacon objects.

#### 5.1.4.7 to_string() [1/2]

```
SDS_API std::string Sds::Falcon::to_string (
            TrackingMode mode )
```

Convert a TrackingMode enum value to a string.

**Parameters**

| | |
|---|---|
| *mode* | TrackingMode enum value. |

**Returns**

string representation of the TrackingMode.

#### 5.1.4.8 to_string() [2/2]

```
SDS_API std::string Sds::Falcon::to_string (
            TrackingModeState state )
```

Convert a TrackingModeState enum value to a string.

**Parameters**

| | |
|---|---|
| *state* | TrackingModeState enum value. |

**Returns**

string representation of the TrackingModeState.

## 5.2 SdsFalcon.h

Go to the documentation of this file.

```
00001 //Copyright © 2017-2024 Six Degrees Space Ltd.  All rights reserved.
00002 //Proprietary and Confidential.  Unauthorized use, disclosure or reproduction is strictly prohibited.
00003
00012 #pragma once
00013 #include <stdint.h>
00014 #include <map>
00015 #include <vector>
00016 #include <string>
00017 #include <memory>
00018 #include <functional>
00019
00020 #ifdef _WIN32
00021     // Windows platform
00022     #ifdef SDS_EXPORTS
00023         #define SDS_API __declspec(dllexport)
00024     #else
00025         #define SDS_API __declspec(dllimport)
00026     #endif
00027 #else
00028     // Non-Windows platform
00029     #define SDS_API __attribute__((visibility("default")))
00030 #endif
00031
00032 #if defined(__GNUC__) || defined(__clang__)
00033 #define DEPRECATED(msg) __attribute__((deprecated(msg)))
00034 #elif defined(_MSC_VER)
00035 #define DEPRECATED(msg) __declspec(deprecated(msg))
00036 #else
00037 #define DEPRECATED(msg)
00038 #endif
00039
00202 namespace Sds {
00203     namespace Falcon {
00204
00211         class SDS_API Version {
00212         public:
```

```
00213             Version(uint8_t major, uint8_t minor, uint8_t patch);
00214
00219             uint8_t getMajor() const;
00220
00225             uint8_t getMinor() const;
00226
00231             uint8_t getPatch() const;
00232
00241             bool isEqual(uint8_t major, uint8_t minor, uint8_t patch) const;
00242
00251             bool isAtLeast(uint8_t major, uint8_t minor, uint8_t patch) const;
00252
00259             std::string getString() const;
00260
00261         private:
00262             uint8_t _major, _minor, _patch;
00263         };
00264
00269         SDS_API Version getVersion();
00270
00277         struct SDS_API NetworkConfig {
00278             uint16_t sensor_id;
00279             std::string sensor_network_name;
00280             uint16_t udp_read_port { 0 };
00281         };
00282
00293         struct SDS_API NetworkAdvancedConfig {
00294             uint16_t sensor_id;
00295             std::string sensor_ip;
00296             uint16_t udp_read_port { 0 };
00297         };
00298
00307         struct SDS_API Beacon {
00308             double x;
00309             double y;
00310             double z;
00311             uint16_t id;
00312         };
00313
00324         struct SDS_API Pose6Dof {
00325             bool valid { false };
00326             double x;
00327             double y;
00328             double z;
00329             double qw;
00330             double qx;
00331             double qy;
00332             double qz;
00333             float var_x, var_y, var_z; // m^2
00334             float var_h, var_p, var_r; // rad^2
00335         };
00336
00344         struct SDS_API PoseRelativeBeacon {
00345             double x;
00346             double y;
00347             double z;
00348             uint16_t id;
00349         };
00350
00360         typedef std::vector<PoseRelativeBeacon> PoseRelativeBeaconCollection;
00361
00362
00370         struct SDS_API RelativeAngle {
00371             double x_angle;
00372             double z_angle;
00373             uint16_t id;
00374             double intensity;
00375             double width;
00376         };
00377
00387         typedef std::vector<RelativeAngle> RelativeAngleCollection;
00388
00395         enum class Severity {
00396             Exception = 1,
00397             Warning,
00398             Informative,
00399         };
00400
00408         enum class CallingLayer {
00409             SdsFalcon = 1,
00410             Algorithm6Dof,
00411             AlgorithmRelativeBeacon,
00412             SdsCommLib,
00413             NetworkConnection,
00414             GyroCalibration,
00415             AlgorithmRelativeAngle,
00416         };
```

```
00417
00424         enum class FalconEventCode {
00425             UnexpectedAlgoInstance,
00426             TrackingModeNotInitalized,
00427             NetworkNotInitalized,
00428             GyroCalibrationRejected,
00429             CommunicationLoopNotClosed,
00430             DidNotReceiveFirmwareInfo,
00431             SensorFirmwareIsNotLatest,
00432             SensorFirmwareIsNotCompatible,
00433         };
00434
00442         struct SDS_API StatusMessage {
00443             Severity severity;
00444             CallingLayer layer;
00445             uint8_t event_code;
00446             std::string message;
00447         };
00448
00455         enum class TrackingMode {
00456             TrackingOFF,
00457             RelativeBeacon,
00458             Sixdof,
00459             RelativeAngle,
00460         };
00461
00468         SDS_API std::string to_string(TrackingMode mode);
00469
00474         enum class TrackingModeState {
00475             Initializing,
00476             Running,
00477             Error,
00478         };
00479
00486         SDS_API std::string to_string(TrackingModeState state);
00487
00495         struct SDS_API Heartbeat {
00496             bool sensor_communication_ok { false };
00497             TrackingMode current_tracking_mode { TrackingMode::TrackingOFF };
00498             TrackingModeState current_tracking_mode_state { TrackingModeState::Initializing };
00499         };
00500
00508         struct SDS_API CurrentShutter {
00509             uint8_t shutter_value;
00510         };
00511
00518         struct SDS_API GyroOffset {
00519             uint16_t sensorId { 0 };
00520             std::string datetime { "" };
00521             int8_t temperature { 0 };
00522             int16_t gyro_x { 0 };
00523             int16_t gyro_y { 0 };
00524             int16_t gyro_z { 0 };
00525             uint16_t standard_dev { 0 };
00526         };
00527
00534         struct SDS_API FieldOfViewReport {
00535             std::map<uint16_t, int> seenBeacons;
00536         };
00537
00554         struct SDS_API ShutterSettings {
00555             uint8_t min_shutter;
00556             uint8_t max_shutter;
00557         };
00558
00563         SDS_API ShutterSettings getShutterSettingsAuto();
00564
00570         SDS_API ShutterSettings getShutterSettingsFixed(uint8_t shutter);
00571
00578         SDS_API ShutterSettings getShutterSettingsRange(uint8_t min_shutter, uint8_t max_shutter);
00579
00587         struct SDS_API Config6Dof {
00588             std::vector<Beacon> map;
00589             std::vector<GyroOffset> gyroCalibration;
00590             bool skip_gyro_calibration { true };
00591             ShutterSettings shutter_settings { getShutterSettingsAuto() };
00592             bool no_gyro_mode { false };
00593             std::string sixdof_dump_path { "" };
00594         };
00595
00607         SDS_API std::vector<Beacon> loadMap(const std::string& path);
00608
00616         struct SDS_API ConfigRelativeBeacon {
00617             ShutterSettings shutter_settings { getShutterSettingsAuto() };
00618             std::vector<uint16_t> expected_beacon_ids;
00619             float field_of_view_cutoff_deg { 45.0 };
00620             std::string rel_lights_dump_path { "" };
```

```
00621         };
00622
00630         struct SDS_API ConfigRelativeAngle {
00631             ShutterSettings shutter_settings { getShutterSettingsAuto() };
00632             std::vector<uint16_t> expected_beacon_ids;
00633             bool matching_mode_single_beacon { false };
00634             float field_of_view_cutoff_deg { 45.0 };
00635             std::string rel_angles_dump_path { "" };
00636         };
00637
00645         struct SDS_API ConfigGyroCalibration {
00646             int duration_seconds { 30 };
00647         };
00648
00656         typedef uint32_t CallbackHandle;
00657
00680         class SDS_API FalconManager {
00681         public:
00682             virtual ~FalconManager() { }
00683
00692             virtual bool initializeNetwork(const NetworkConfig&) = 0;
00693
00702             virtual bool initializeNetwork(const NetworkAdvancedConfig&) = 0;
00703
00712             virtual bool initialize6Dof(const Config6Dof&) = 0;
00713
00722             virtual bool initializeRelativeBeacon(const ConfigRelativeBeacon&) = 0;
00723
00732             virtual bool initializeRelativeAngle(const ConfigRelativeAngle&) = 0;
00733
00739             virtual void setTrackingMode(const TrackingMode&) = 0;
00740
00744             virtual Version getFirmwareVersion() = 0;
00745
00755             virtual std::vector<GyroOffset> getGyroCalibration(const ConfigGyroCalibration&) = 0;
00756
00757             virtual Pose6Dof getPose6Dof() = 0;
00758
00759             virtual PoseRelativeBeaconCollection getPoseRelativeBeaconCollection() = 0;
00760
00761             virtual RelativeAngleCollection getRelativeAngleCollection() = 0;
00762
00763             virtual StatusMessage getStatusMessage() = 0;
00764
00765             virtual FieldOfViewReport getFieldOfViewReport() = 0;
00766
00767             virtual CurrentShutter getCurrentShutter() = 0;
00768
00769             virtual CallbackHandle registerPose6DofCallback(const std::function<void(Pose6Dof)>&) = 0;
00770
00771             virtual CallbackHandle registerPoseRelativeBeaconCallback(const
      std::function<void(PoseRelativeBeaconCollection)>&) = 0;
00772
00773             virtual CallbackHandle registerRelativeAngleCallback(const
      std::function<void(RelativeAngleCollection)>&) = 0;
00774
00775             virtual CallbackHandle registerMessageCallback(const std::function<void(StatusMessage)>&)
      = 0;
00776
00777             virtual CallbackHandle registerFieldOfViewReportCallback(const
      std::function<void(FieldOfViewReport)>&) = 0;
00778
00779             virtual CallbackHandle registerHeartbeatCallback(const std::function<void(Heartbeat)>&) =
      0;
00780
00781             virtual CallbackHandle registerCurrentShutterCallback(const
      std::function<void(CurrentShutter)>&) = 0;
00782
00783             virtual void removePose6DofCallback(const CallbackHandle&) = 0;
00784
00785             virtual void removePoseRelativeBeaconCallback(const CallbackHandle&) = 0;
00786
00787             virtual void removeRelativeAngleCallback(const CallbackHandle&) = 0;
00788
00789             virtual void removeMessageCallback(const CallbackHandle&) = 0;
00790
00791             virtual void removeFieldOfViewCallback(const CallbackHandle&) = 0;
00792
00793             virtual void removeHeartbeatCallback(const CallbackHandle&) = 0;
00794
00795             virtual void removeCurrentShutterCallback(const CallbackHandle&) = 0;
00796
00805             virtual void swapMap(const std::vector<Beacon>& map) = 0;
00806         };
00807
00816         SDS_API std::unique_ptr<FalconManager> createFalconManager();
00817     }
```

```
00818 }
```