

Motorcycle Awareness System (MAS)

ME553 Group EPE 6

Generated by Doxygen 1.8.6

Fri Apr 17 2015 23:53:22

Contents

1	Motorcycle Awareness System Overview	1
2	Todo List	2
3	Class Index	2
3.1	Class List	2
4	File Index	2
4.1	File List	2
5	Class Documentation	2
5.1	BluetoothMessage_t Struct Reference	3
5.1.1	Detailed Description	3
5.1.2	Member Data Documentation	3
5.2	CanSignal_t Struct Reference	4
5.2.1	Detailed Description	4
5.2.2	Member Data Documentation	4
5.3	GpsSignal_t Struct Reference	5
5.3.1	Detailed Description	6
5.3.2	Member Data Documentation	6
5.4	MotorcycleAwarenessSystem Class Reference	6
5.4.1	Detailed Description	8
5.4.2	Constructor & Destructor Documentation	8
5.4.3	Member Function Documentation	9
5.4.4	Member Data Documentation	13
5.5	MotorCycleLocation_t Struct Reference	14
5.5.1	Detailed Description	15
5.5.2	Member Data Documentation	15
5.6	RadarSignal_t Struct Reference	15
5.6.1	Detailed Description	16
5.6.2	Member Data Documentation	16
6	File Documentation	17
6.1	main.cpp File Reference	17
6.1.1	Function Documentation	17
6.2	main.cpp	18
6.3	mainpage.dox File Reference	19

6.4	MotorcycleAwarenessSystem.cpp File Reference	19
6.5	MotorcycleAwarenessSystem.cpp	19
6.6	MotorcycleAwarenessSystem.hpp File Reference	21
6.7	MotorcycleAwarenessSystem.hpp	22
6.8	MotorcycleAwarenessSystemTypes.hpp File Reference	23
6.8.1	Typedef Documentation	25
6.8.2	Enumeration Type Documentation	25
6.9	MotorcycleAwarenessSystemTypes.hpp	25

Index

27

1 Motorcycle Awareness System Overview

This low-level design documentation details a mockup of the Motorcycle Awareness System. It provides part of the functionality of the MAS for both the motorcycle and car. Mocking of input signals such as the radar signals, vehicle-to-vehicle (V2V) communication signals, and GPS signals mimics the interactions that the MAS would have with actual sensor data on the finished product.

The implemented functionality includes continuous tracking of the motorcycle using GPS, determining whether a hazard is present using the motorcycle's radar sensor signals, and relaying a warning to the motorcycle rider of a potential threat. For the car, the logic determines whether the car's blinker is on based on blinker signal data on the CAN bus, determines whether a motorcycle is in range and assess the potential danger, and issues a warning to the driver as necessary.

MAS Concept Sketch

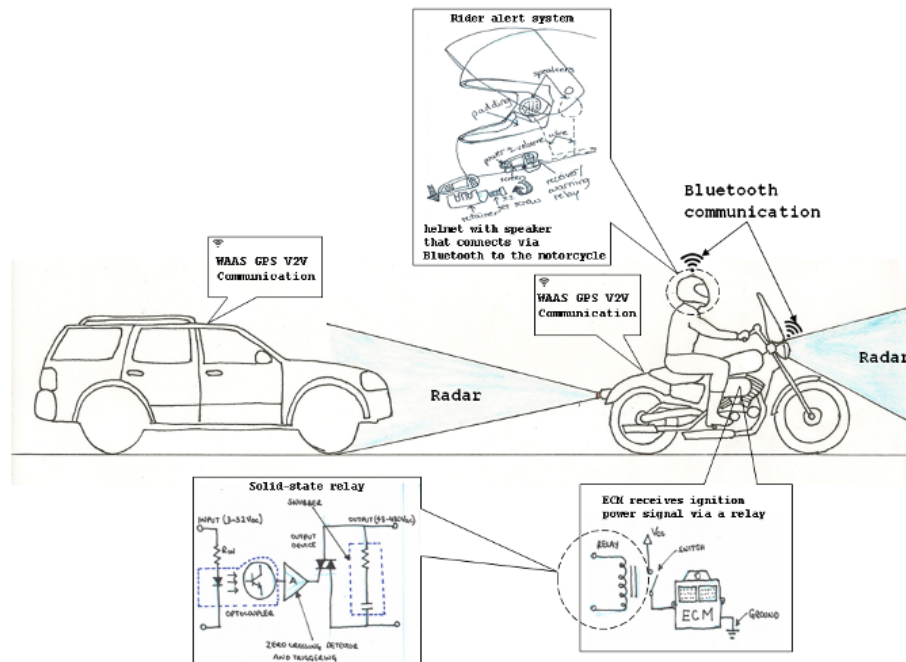


Figure 1: MAS system overview

2 Todo List

Member [MotorcycleAwarenessSystem::GetMotorcycleLocation](#) (void)

Acquire the data packet from the motorcycle

Member [MotorcycleAwarenessSystem::IsMotorcycleInRange](#) (void)

Process the data packet and determine threat

Member [MotorcycleAwarenessSystem::RelayWarningToOperator](#) (void)

Transmit the bluetooth message to the operator

3 Class Index

3.1 Class List

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

BluetoothMessage_t	
Struct for bluetooth message	3
CanSignal_t	
Structure that emulates a CAN bus signal	4
GpsSignal_t	
Structure that emulates a GPS signal	5
MotorcycleAwarenessSystem	
Class declaration for the Motorcycle Awareness System (MAS)	6
MotorCycleLocation_t	
Struct for the V2V data	14
RadarSignal_t	
Structure that emulates a Radar signal	15

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

main.cpp	17
MotorcycleAwarenessSystem.cpp	19
MotorcycleAwarenessSystem.hpp	21
MotorcycleAwarenessSystemTypes.hpp	23

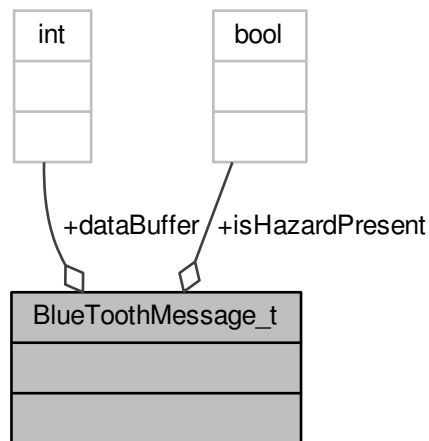
5 Class Documentation

5.1 BluetoothMessage_t Struct Reference

Struct for bluetooth message.

```
#include <MotorcycleAwarenessSystemTypes.hpp>
```

Collaboration diagram for BluetoothMessage_t:



Public Attributes

- bool `isHazardPresent`
Hazard flag.
- unsigned int `dataBuffer` [255]
Bluetooth data buffer.

5.1.1 Detailed Description

Struct for bluetooth message.

Definition at line 41 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.1.2 Member Data Documentation

5.1.2.1 unsigned int BluetoothMessage_t::dataBuffer[255]

Bluetooth data buffer.

Definition at line 44 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.1.2.2 bool BluetoothMessage_t::isHazardPresent

Hazard flag.

Definition at line 43 of file [MotorcycleAwarenessSystemTypes.hpp](#).

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

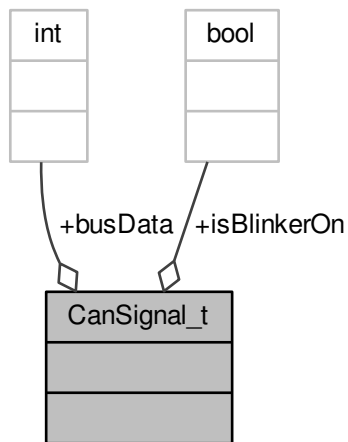
- [MotorcycleAwarenessSystemTypes.hpp](#)

5.2 CanSignal_t Struct Reference

Structure that emulates a CAN bus signal.

```
#include <MotorcycleAwarenessSystemTypes.hpp>
```

Collaboration diagram for CanSignal_t:



Public Attributes

- bool [isBlinkerOn](#)
Blinker signal.
- int [busData](#) [16]
Other CAN bus data.

5.2.1 Detailed Description

Structure that emulates a CAN bus signal.

Definition at line 15 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.2.2 Member Data Documentation

5.2.2.1 int CanSignal_t::busData[16]

Other CAN bus data.

Definition at line 18 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.2.2.2 bool CanSignal_t::isBlinkerOn

Blinker signal.

Definition at line 17 of file [MotorcycleAwarenessSystemTypes.hpp](#).

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

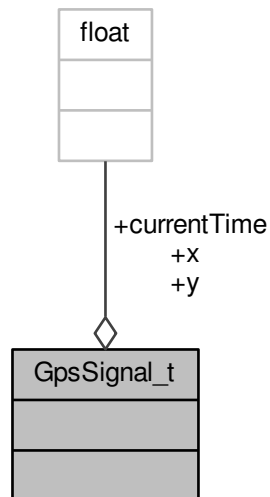
- [MotorcycleAwarenessSystemTypes.hpp](#)

5.3 GpsSignal_t Struct Reference

Structure that emulates a GPS signal.

```
#include <MotorcycleAwarenessSystemTypes.hpp>
```

Collaboration diagram for GpsSignal_t:



Public Attributes

- [Coordinate_t x](#)
x-axis coordinate
- [Coordinate_t y](#)
y-axis coordinate
- [currentTime_t currentTime](#)

Current time at coordinates x,y.

5.3.1 Detailed Description

Structure that emulates a GPS signal.

Definition at line 24 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.3.2 Member Data Documentation

5.3.2.1 `currentTime_t GpsSignal_t::currentTime`

Current time at coordinates x,y.

Definition at line 28 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.3.2.2 `Coordinate_t GpsSignal_t::x`

x-axis coordinate

Definition at line 26 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.3.2.3 `Coordinate_t GpsSignal_t::y`

y-axis coordinate

Definition at line 27 of file [MotorcycleAwarenessSystemTypes.hpp](#).

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

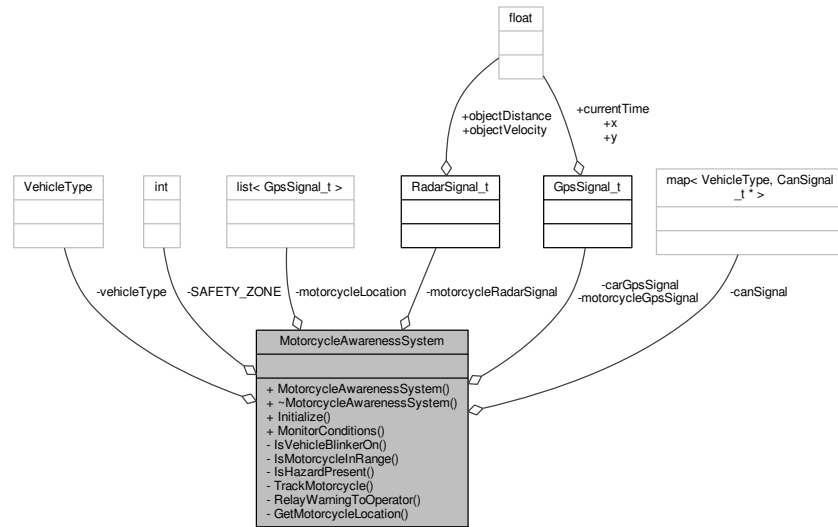
- [MotorcycleAwarenessSystemTypes.hpp](#)

5.4 MotorcycleAwarenessSystem Class Reference

Class declaration for the Motorcycle Awareness System (MAS)

```
#include <MotorcycleAwarenessSystem.hpp>
```


Collaboration diagram for MotorcycleAwarenessSystem:



Public Member Functions

- [MotorcycleAwarenessSystem \(VehicleType vehicleType\)](#)
Constructor.
- [~MotorcycleAwarenessSystem \(void\)](#)
Destructor.
- void [Initialize \(CanSignal_t *motorcycleCanSignal, CanSignal_t *carCanSignal, RadarSignal_t *motorcycleRadarSignal, GpsSignal_t *motorcycleGpsSignal, GpsSignal_t *carGpsSignal\)](#)
Method to initialize the MAS system.
- void [MonitorConditions \(void\)](#)
Method to continuously monitor the conditions during run-time.

Private Member Functions

- bool [IsVehicleBlinkerOn \(void\)](#)
Method to determine whether the car's blinker is ON.
- bool [IsMotorcycleInRange \(void\)](#)
Method to determine whether the motorcycle is within the car's range.
- bool [IsHazardPresent \(void\)](#)
- void [TrackMotorcycle \(void\)](#)
Method to track the motorcycle using its GPS signal.
- void [RelayWarningToOperator \(void\)](#)
Method to relay a warning to operator via bluetooth connectivity.
- [MotorCycleLocation_t GetMotorcycleLocation \(void\)](#)

Private Attributes

- `std::list< GpsSignal_t > motorcycleLocation`
Container used to track the motorcycle's location.
- `VehicleType vehicleType`
The vehicle type (motorcycle or car)
- `std::map< VehicleType,
CanSignal_t * > canSignal`
Storage for the CAN signals.
- `RadarSignal_t * motorcycleRadarSignal`
Pointer to a motorcycle radar signal.
- `GpsSignal_t * motorcycleGpsSignal`
Pointer to a motorcycle GPS signal.
- `GpsSignal_t * carGpsSignal`
Pointer to a car GPS signal.

Static Private Attributes

- `static const unsigned int SAFETY_ZONE = 15U`

5.4.1 Detailed Description

Class declaration for the Motorcycle Awareness System (MAS)

Definition at line 12 of file [MotorcycleAwarenessSystem.hpp](#).

5.4.2 Constructor & Destructor Documentation

5.4.2.1 `MotorcycleAwarenessSystem::MotorcycleAwarenessSystem (VehicleType vehicleType)`

Constructor.

Class definition for the Motorcycle Awareness System (MAS). This class processes various signals and interactions to realize the MAS

Definition at line 13 of file [MotorcycleAwarenessSystem.cpp](#).

```
00014     :vehicleType( vehicleType )
00015 {
00016     // Do nothing
00017 }
```

5.4.2.2 `MotorcycleAwarenessSystem::~MotorcycleAwarenessSystem (void)`

Destructor.

Definition at line 20 of file [MotorcycleAwarenessSystem.cpp](#).

```
00021 {
00022     // Do nothing
00023 }
```

5.4.3 Member Function Documentation

5.4.3.1 `MotorCycleLocation_t MotorcycleAwarenessSystem::GetMotorcycleLocation (void)` [private]

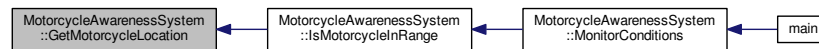
Todo Acquire the data packet from the motorcycle

Definition at line 129 of file [MotorcycleAwarenessSystem.cpp](#).

```

00130 {
00131     // Dummy motorcycle location from V2V communication
00132     MotorCycleLocation_t motorCycleLocation;
00133
00134
00135     return motorCycleLocation;
00136 }
```

Here is the caller graph for this function:

5.4.3.2 `void MotorcycleAwarenessSystem::Initialize (CanSignal_t * motorcycleCanSignal, CanSignal_t * carCanSignal, RadarSignal_t * motorcycleRadarSignal, GpsSignal_t * motorcycleGpsSignal, GpsSignal_t * carGpsSignal)`

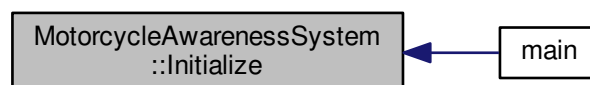
Method to initialize the MAS system.

Definition at line 26 of file [MotorcycleAwarenessSystem.cpp](#).

```

00029 {
00030     // Initialize the motorcycle radar signal
00031     this->motorcycleRadarSignal = motorcycleRadarSignal;
00032
00033     // Initialize the canSignal map
00034     canSignal[MOTORCYCLE] = motorcycleCanSignal;
00035     canSignal[CAR] = carCanSignal;
00036
00037     // Initialize the GPS signals
00038     this->motorcycleGpsSignal = motorcycleGpsSignal;
00039     this->motorcycleGpsSignal = motorcycleGpsSignal;
00040 }
```

Here is the caller graph for this function:



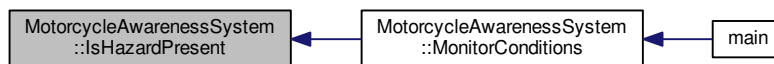
5.4.3.3 bool MotorcycleAwarenessSystem::IsHazardPresent (void) [private]

Method to determine whether a hazard is within the motorcycle's safety zone. An object within the safety zone is of potential danger to the motorcycle rider.

Definition at line 97 of file [MotorcycleAwarenessSystem.cpp](#).

```
00098 {
00099     bool isHazardPresent = false;
00100
00101     if ( abs( this->motorcycleRadarSignal->objectDistance ) >=
SAFETY_ZONE )
00102     {
00103         isHazardPresent = true;
00104     }
00105
00106     return isHazardPresent;
00107 }
```

Here is the caller graph for this function:



5.4.3.4 bool MotorcycleAwarenessSystem::IsMotorcycleInRange (void) [private]

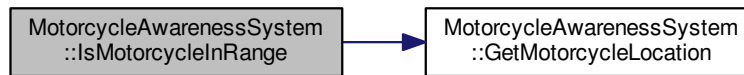
Method to determine whether the motorcycle is within the car's range.

Todo Process the data packet and determine threat

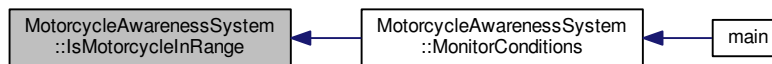
Definition at line 75 of file [MotorcycleAwarenessSystem.cpp](#).

```
00076 {
00077     bool isInRange = false;
00078
00079     // Determine where the motorcycle is relative to the car using the GPS signals
00080     if ( ((abs( this->carGpsSignal->x ) + SAFETY_ZONE) >= abs( this->
motorcycleGpsSignal->x )) ||
00081         ((abs( this->carGpsSignal->y ) + SAFETY_ZONE) >= abs( this->
motorcycleGpsSignal->y )))
00082     {
00083         isInRange = true;
00084     }
00085     else
00086     {
00087         // Analyze the V2V data for a threat
00088         MotorcycleLocation_t motorCycleLocation =
GetMotorcycleLocation();
00090     }
00091
00092     return isInRange;
00093 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.3.5 `bool MotorcycleAwarenessSystem::IsVehicleBlinkerOn (void) [private]`

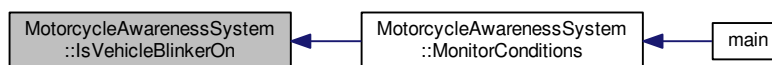
Method to determine whether the car's blinker is ON.

Definition at line 69 of file [MotorcycleAwarenessSystem.cpp](#).

```

00070 {
00071     return (this->canSignal[vehicleType])>isBlinkerOn;
00072 }
  
```

Here is the caller graph for this function:



5.4.3.6 `void MotorcycleAwarenessSystem::MonitorConditions (void)`

Method to continuously monitor the conditions during run-time.

Definition at line 43 of file [MotorcycleAwarenessSystem.cpp](#).

```

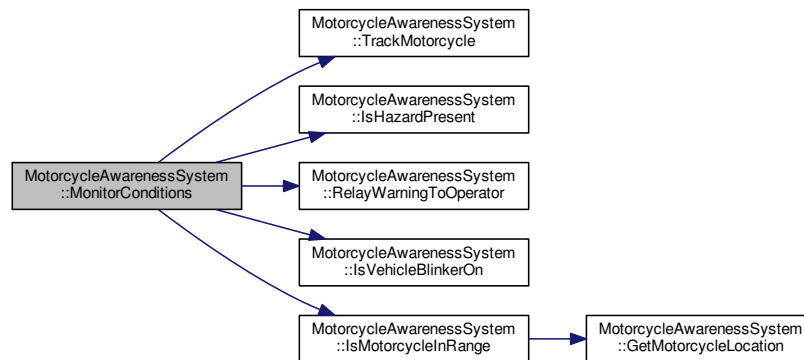
00044 {
00045     if ( MOTORCYCLE == vehicleType )
00046     {
00047         // Track the motorcycle
00048         TrackMotorcycle();
00049         // Check for hazards
00050         if ( IsHazardPresent() == true )
  
```

```

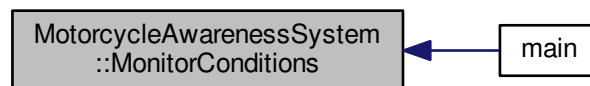
00051      {
00052          // Warn the motorcycle operator
00053          RelayWarningToOperator();
00054      }
00055  }
00056  // vehicleType == CAR
00057  else
00058  {
00059      // Check for hazards
00060      if ( (IsVehicleBlinkerOn() == true) && (
IsMotorcycleInRange() == true) )
00061      {
00062          // Relay message to the car driver
00063          RelayWarningToOperator();
00064      }
00065  }
00066  }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.3.7 void MotorcycleAwarenessSystem::RelayWarningToOperator (void) [private]

Method to relay a warning to operator via bluetooth connectivity.

Todo Transmit the bluetooth message to the operator

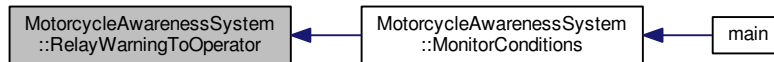
Definition at line 117 of file `MotorcycleAwarenessSystem.cpp`.

```

00118 {
00119     // Assemble the message to be sent
00120     BluetoothMessage_t blueToothMessage;
00121     blueToothMessage.isHazardPresent = true;
00122     // Dummy message
00123     blueToothMessage.dataBuffer[0] = 0x2015;
00124
00126 }

```

Here is the caller graph for this function:



5.4.3.8 void MotorcycleAwarenessSystem::TrackMotorcycle(void) [private]

Method to track the motorcycle using its GPS signal.

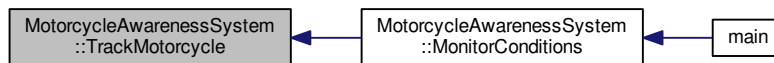
Definition at line 110 of file [MotorcycleAwarenessSystem.cpp](#).

```

00111 {
00112     // Push the motorcycle's GPS location onto the list
00113     motorcycleLocation.push_front( *motorcycleGpsSignal );
00114 }

```

Here is the caller graph for this function:



5.4.4 Member Data Documentation

5.4.4.1 std::map<VehicleType, CanSignal_t*> MotorcycleAwarenessSystem::canSignal [private]

Storage for the CAN signals.

Definition at line 27 of file [MotorcycleAwarenessSystem.hpp](#).

5.4.4.2 GpsSignal_t* MotorcycleAwarenessSystem::carGpsSignal [private]

Pointer to a car GPS signal.

Definition at line 30 of file [MotorcycleAwarenessSystem.hpp](#).

5.4.4.3 GpsSignal_t* MotorcycleAwarenessSystem::motorcycleGpsSignal [private]

Pointer to a motorcycle GPS signal.

Definition at line 29 of file [MotorcycleAwarenessSystem.hpp](#).

5.4.4.4 `std::list<GpsSignal_t> MotorcycleAwarenessSystem::motorcycleLocation` [private]

Container used to track the motorcycle's location.

Definition at line 23 of file [MotorcycleAwarenessSystem.hpp](#).

5.4.4.5 `RadarSignal_t* MotorcycleAwarenessSystem::motorcycleRadarSignal` [private]

Pointer to a motorcycle radar signal.

Definition at line 28 of file [MotorcycleAwarenessSystem.hpp](#).

5.4.4.6 `const unsigned int MotorcycleAwarenessSystem::SAFETY_ZONE = 15U` [static], [private]

Distance from the radar sensor in feet in which a detected object becomes a potential danger

Definition at line 24 of file [MotorcycleAwarenessSystem.hpp](#).

5.4.4.7 `VehicleType MotorcycleAwarenessSystem::vehicleType` [private]

The vehicle type (motorcycle or car)

Definition at line 26 of file [MotorcycleAwarenessSystem.hpp](#).

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

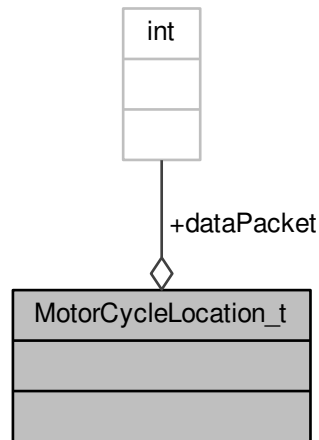
- [MotorcycleAwarenessSystem.hpp](#)
- [MotorcycleAwarenessSystem.cpp](#)

5.5 MotorCycleLocation_t Struct Reference

Struct for the V2V data.

```
#include <MotorcycleAwarenessSystemTypes.hpp>
```


Collaboration diagram for MotorcycleLocation_t:



Public Attributes

- [DataPacket_t dataPacket](#)
V2V communication data packet.

5.5.1 Detailed Description

Struct for the V2V data.

Definition at line 49 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.5.2 Member Data Documentation

5.5.2.1 DataPacket_t MotorcycleLocation_t::dataPacket

V2V communication data packet.

Definition at line 51 of file [MotorcycleAwarenessSystemTypes.hpp](#).

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

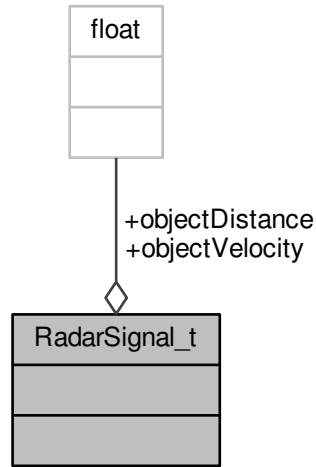
- [MotorcycleAwarenessSystemTypes.hpp](#)

5.6 RadarSignal_t Struct Reference

Structure that emulates a Radar signal.

```
#include <MotorcycleAwarenessSystemTypes.hpp>
```

Collaboration diagram for RadarSignal_t:



Public Attributes

- [ObjectDistance](#) `objectDistance`
Distance from sensed object to radar sensor.
- [ObjectVelocity](#) `objectVelocity`
Velocity of sensed object.

5.6.1 Detailed Description

Structure that emulates a Radar signal.

Definition at line 34 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.6.2 Member Data Documentation

5.6.2.1 ObjectDistance RadarSignal_t::objectDistance

Distance from sensed object to radar sensor.

Definition at line 36 of file [MotorcycleAwarenessSystemTypes.hpp](#).

5.6.2.2 ObjectVelocity RadarSignal_t::objectVelocity

Velocity of sensed object.

Definition at line 37 of file [MotorcycleAwarenessSystemTypes.hpp](#).

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

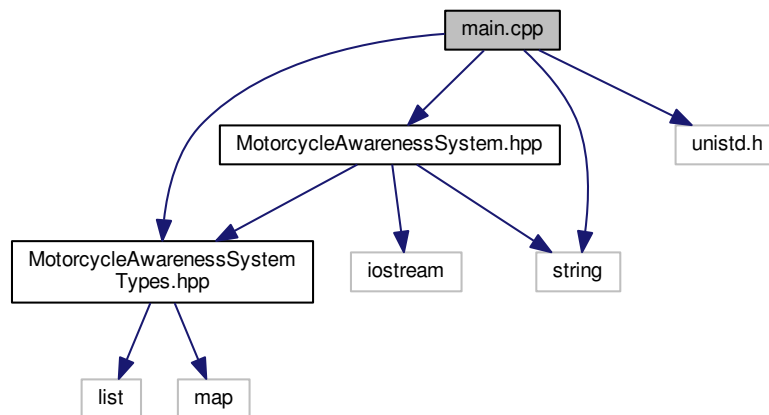
- [MotorcycleAwarenessSystemTypes.hpp](#)

6 File Documentation

6.1 main.cpp File Reference

```
#include "MotorcycleAwarenessSystem.hpp"
#include "MotorcycleAwarenessSystemTypes.hpp"
#include <string>
#include <unistd.h>
```

Include dependency graph for main.cpp:



Functions

- `int main (void)`

6.1.1 Function Documentation

6.1.1.1 `int main (void)`

Definition at line 10 of file [main.cpp](#).

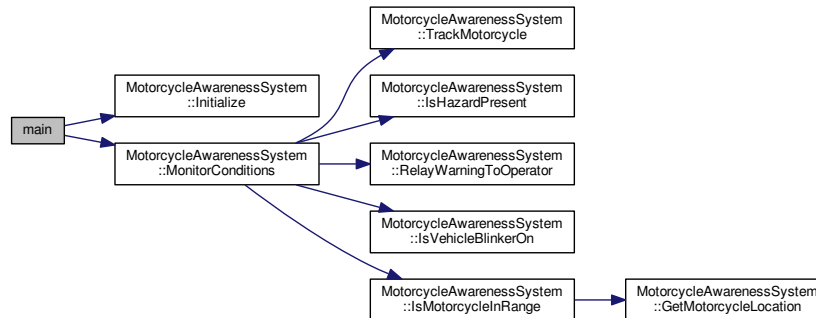
```
00011 {
00012     // Define mock signals
00013     CanSignal_t* motorcycleCanSignal = new CanSignal_t;
00014     CanSignal_t* carCanSignal = new CanSignal_t;
00015     RadarSignal_t* motorcycleRadarSignal = new RadarSignal_t;
00016     GpsSignal_t* motorcycleGpsSignal = new GpsSignal_t;
00017     GpsSignal_t* carGpsSignal = new GpsSignal_t;;
00018
00019     // Instantiate the Motorcycle Awareness System (MAS) for the motorcycle
00020     MotorcycleAwarenessSystem MAS_motorcycle(
        MOTORCYCLE );
00021     // Initialize the car MAS
00022     MAS_motorcycle.Initialize( motorcycleCanSignal, carCanSignal, motorcycleRadarSignal,
```

```

00023                                     motorcycleGpsSignal, carGpsSignal );
00024
00025     // Instantiate the Motorcycle Awareness System (MAS) for the car
00026     MotorcycleAwarenessSystem MAS_car( CAR );
00027     // Initialize the car MAS
00028     MAS_car.Initialize( motorcycleCanSignal, carCanSignal, motorcycleRadarSignal,
00029                                     motorcycleGpsSignal, carGpsSignal );
00030
00031     // Run the MAS systems for the car & motorcycle
00032     while ( true )
00033     {
00034         MAS_motorcycle.MonitorConditions();
00035         MAS_car.MonitorConditions();
00036         usleep(250);
00037
00038     #ifdef MAS_DEBUG
00039         std::cout << "The MAS is running" << std::endl;
00040     #endif
00041     }
00042
00043     return 0;
00044 }

```

Here is the call graph for this function:



6.2 main.cpp

```

00001 #include "MotorcycleAwarenessSystem.hpp"
00002 #include "MotorcycleAwarenessSystemTypes.hpp"
00003
00004 #ifdef MAS_DEBUG
00005 #include <iostream>
00006 #endif
00007 #include <string>
00008 #include <unistd.h>
00009
00010 int main( void )
00011 {
00012     // Define mock signals
00013     CanSignal_t* motorcycleCanSignal = new CanSignal_t;
00014     CanSignal_t* carCanSignal = new CanSignal_t;
00015     RadarSignal_t* motorcycleRadarSignal = new RadarSignal_t;
00016     GpsSignal_t* motorcycleGpsSignal = new GpsSignal_t;
00017     GpsSignal_t* carGpsSignal = new GpsSignal_t;
00018
00019     // Instantiate the Motorcycle Awareness System (MAS) for the motorcycle
00020     MotorcycleAwarenessSystem MAS_motorcycle(
00021         MOTORCYCLE );
00022     // Initialize the car MAS
00023     MAS_motorcycle.Initialize( motorcycleCanSignal, carCanSignal, motorcycleRadarSignal,
00024                                     motorcycleGpsSignal, carGpsSignal );
00025
00026     // Instantiate the Motorcycle Awareness System (MAS) for the car
00027     MotorcycleAwarenessSystem MAS_car( CAR );

```

```

00027 // Initialize the car MAS
00028 MAS_car.Initialize( motorcycleCanSignal, carCanSignal, motorcycleRadarSignal,
00029                   motorcycleGpsSignal, carGpsSignal );
00030
00031 // Run the MAS systems for the car & motorcycle
00032 while ( true )
00033 {
00034     MAS_motorcycle.MonitorConditions();
00035     MAS_car.MonitorConditions();
00036     usleep(250);
00037 }
00038 #ifdef MAS_DEBUG
00039     std::cout << "The MAS is running" << std::endl;
00040 #endif
00041 }
00042
00043 return 0;
00044 }

```

6.3 mainpage.dox File Reference

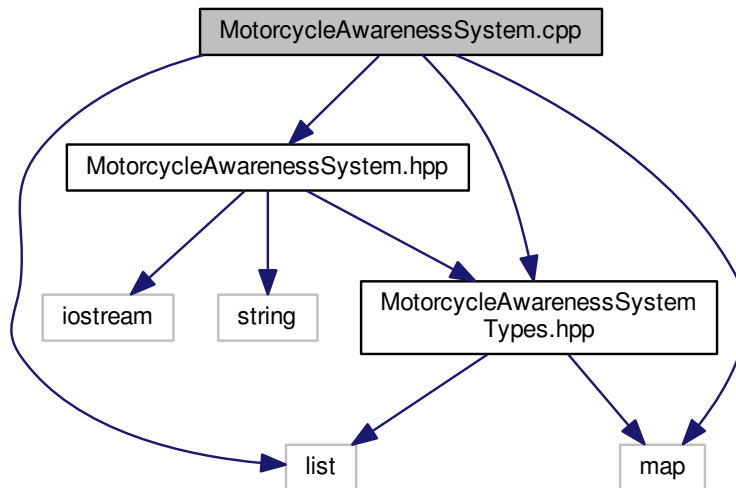
6.4 MotorcycleAwarenessSystem.cpp File Reference

```

#include "MotorcycleAwarenessSystem.hpp"
#include "MotorcycleAwarenessSystemTypes.hpp"
#include <list>
#include <map>

```

Include dependency graph for MotorcycleAwarenessSystem.cpp:



6.5 MotorcycleAwarenessSystem.cpp

```

00001 #include "MotorcycleAwarenessSystem.hpp"
00007 #include "MotorcycleAwarenessSystemTypes.hpp"
00008
00009 #include <list>
00010 #include <map>

```

```

00011
00013 MotorcycleAwarenessSystem::MotorcycleAwarenessSystem(
    VehicleType vehicleType )
00014     :vehicleType( vehicleType )
00015 {
00016     // Do nothing
00017 }
00018
00020 MotorcycleAwarenessSystem::~MotorcycleAwarenessSystem(
    void )
00021 {
00022     // Do nothing
00023 }
00024
00026 void MotorcycleAwarenessSystem::Initialize(
    CanSignal_t* motorcycleCanSignal, CanSignal_t* carCanSignal,
00027     RadarSignal_t* motorcycleRadarSignal,
    GpsSignal_t* motorcycleGpsSignal,
00028     GpsSignal_t* carGpsSignal )
00029 {
00030     // Initialize the motorcycle radar signal
00031     this->motorcycleRadarSignal = motorcycleRadarSignal;
00032
00033     // Initialize the canSignal map
00034     canSignal[MOTORCYCLE] = motorcycleCanSignal;
00035     canSignal[CAR] = carCanSignal;
00036
00037     // Initialize the GPS signals
00038     this->motorcycleGpsSignal = motorcycleGpsSignal;
00039     this->motorcycleGpsSignal = motorcycleGpsSignal;
00040 }
00041
00043 void MotorcycleAwarenessSystem::MonitorConditions( void )
00044 {
00045     if ( MOTORCYCLE == vehicleType )
00046     {
00047         // Track the motorcycle
00048         TrackMotorcycle();
00049         // Check for hazards
00050         if ( IsHazardPresent() == true )
00051         {
00052             // Warn the motorcycle operator
00053             RelayWarningToOperator();
00054         }
00055     }
00056     // vehicleType == CAR
00057     else
00058     {
00059         // Check for hazards
00060         if ( (IsVehicleBlinkerOn() == true) && (
            IsMotorcycleInRange() == true) )
00061         {
00062             // Relay message to the car driver
00063             RelayWarningToOperator();
00064         }
00065     }
00066 }
00067
00069 bool MotorcycleAwarenessSystem::IsVehicleBlinkerOn( void )
00070 {
00071     return (this->canSignal[vehicleType])>isBlinkerOn;
00072 }
00073
00075 bool MotorcycleAwarenessSystem::IsMotorcycleInRange( void )
00076 {
00077     bool isInRange = false;
00078
00079     // Determine where the motorcycle is relative to the car using the GPS signals
00080     if ( ((abs( this->carGpsSignal->x ) + SAFETY_ZONE) >= abs( this->
        motorcycleGpsSignal->x )) ||
00081         ((abs( this->carGpsSignal->y ) + SAFETY_ZONE) >= abs( this->
        motorcycleGpsSignal->y )))
00082     {
00083         isInRange = true;
00084     }
00085     else
00086     {
00087         // Analyze the V2V data for a threat
00088         MotorCycleLocation_t motorCycleLocation =
            GetMotorcycleLocation();
00090     }

```

```

00091
00092     return isInRange;
00093 }
00094
00097 bool MotorcycleAwarenessSystem::IsHazardPresent( void )
00098 {
00099     bool isHazardPresent = false;
00100
00101     if ( abs( this->motorcycleRadarSignal->objectDistance ) >=
SAFETY_ZONE )
00102     {
00103         isHazardPresent = true;
00104     }
00105
00106     return isHazardPresent;
00107 }
00108
00110 void MotorcycleAwarenessSystem::TrackMotorcycle( void )
00111 {
00112     // Push the motorcycle's GPS location onto the list
00113     motorcycleLocation.push_front( *motorcycleGpsSignal );
00114 }
00115
00117 void MotorcycleAwarenessSystem::RelayWarningToOperator(
void )
00118 {
00119     // Assemble the message to be sent
00120     BlueToothMessage_t blueToothMessage;
00121     blueToothMessage.isHazardPresent = true;
00122     // Dummy message
00123     blueToothMessage.dataBuffer[0] = 0x2015;
00124
00126 }
00127
00128 // Method to Get motorcycle's location via V2V communication
00129 MotorCycleLocation_t
MotorcycleAwarenessSystem::GetMotorcycleLocation( void )
00130 {
00131     // Dummy motorcycle location from V2V communication
00132     MotorCycleLocation_t motorCycleLocation;
00133
00134     return motorCycleLocation;
00135 }
00136 }

```

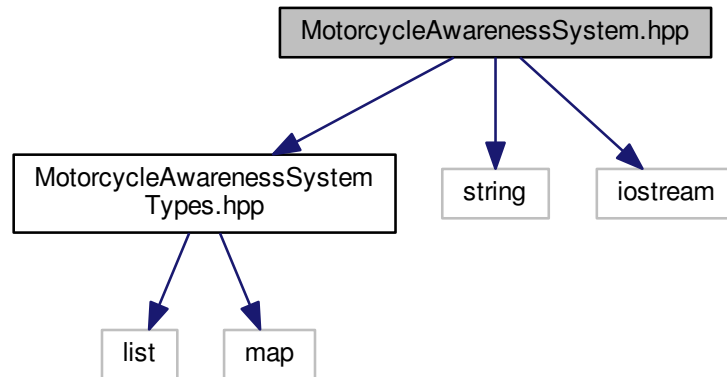
6.6 MotorcycleAwarenessSystem.hpp File Reference

```

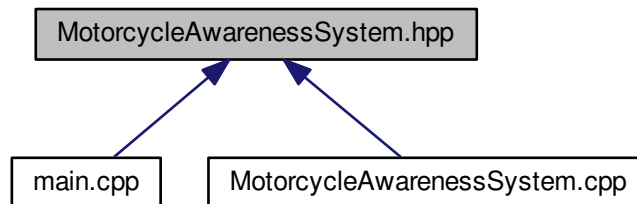
#include "MotorcycleAwarenessSystemTypes.hpp"
#include <string>
#include <iostream>

```

Include dependency graph for MotorcycleAwarenessSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [MotorcycleAwarenessSystem](#)
Class declaration for the Motorcycle Awareness System (MAS)

6.7 MotorcycleAwarenessSystem.hpp

```

00001 #ifndef MOTORCYCLEAWARENESSSYSTEM_HPP
00005 #define MOTORCYCLEAWARENESSSYSTEM_HPP
00006
00007 #include "MotorcycleAwarenessSystemTypes.hpp"
00008
00009 #include <string>
00010 #include <iostream>
00011

```



```

00012 class MotorcycleAwarenessSystem
00013 {
00014     public:
00015         MotorcycleAwarenessSystem( VehicleType
vehicleType );
00016         ~MotorcycleAwarenessSystem( void );
00017         void Initialize( CanSignal_t* motorcycleCanSignal,
CanSignal_t* carCanSignal,
00018             RadarSignal_t* motorcycleRadarSignal,
GpsSignal_t* motorcycleGpsSignal,
00019             GpsSignal_t* carGpsSignal );
00020         void MonitorConditions( void );
00021     private:
00022         std::list<GpsSignal_t> motorcycleLocation;
00023         static const unsigned int SAFETY_ZONE = 15U;
00024         VehicleType vehicleType;
00025         std::map<VehicleType, CanSignal_t*> canSignal;
00026         RadarSignal_t* motorcycleRadarSignal;
00027         GpsSignal_t* motorcycleGpsSignal;
00028         GpsSignal_t* carGpsSignal;
00029         bool IsVehicleBlinkerOn( void );
00030         bool IsMotorcycleInRange( void );
00031         bool IsHazardPresent( void );
00032         void TrackMotorcycle( void );
00033         void RelayWarningToOperator( void );
00034         MotorcycleLocation_t GetMotorcycleLocation( void );
00035 };
00036 #endif

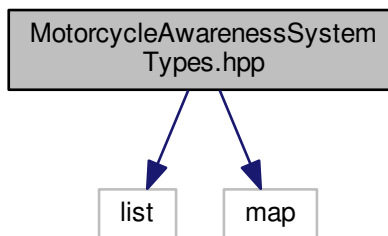
```

6.8 MotorcycleAwarenessSystemTypes.hpp File Reference

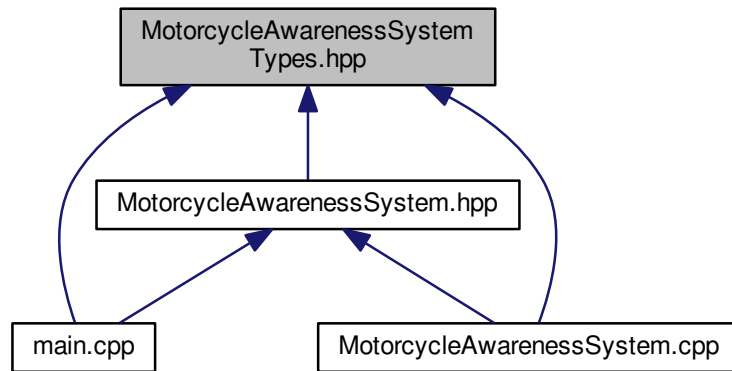
```
#include <list>
```

```
#include <map>
```

Include dependency graph for MotorcycleAwarenessSystemTypes.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [CanSignal_t](#)
Structure that emulates a CAN bus signal.
- struct [GpsSignal_t](#)
Structure that emulates a GPS signal.
- struct [RadarSignal_t](#)
Structure that emulates a Radar signal.
- struct [BluetoothMessage_t](#)
Struct for bluetooth message.
- struct [MotorCycleLocation_t](#)
Struct for the V2V data.

Typedefs

- typedef float [Coordinate_t](#)
GPS coordinates.
- typedef float [currentTime_t](#)
- typedef float [ObjectDistance](#)
Distance of an object from a detecting radar sensor.
- typedef float [ObjectVelocity](#)
- typedef int [DataPacket_t](#) [255]

Enumerations

- enum [VehicleType](#) { [MOTORCYCLE](#), [CAR](#) }
Enum for vehicle type.

6.8.1 Typedef Documentation

6.8.1.1 typedef float **Coordinate_t**

GPS coordinates.

Definition at line 21 of file [MotorcycleAwarenessSystemTypes.hpp](#).

6.8.1.2 typedef float **currentTime_t**

Current time for a pair of GPS coordinates

Definition at line 22 of file [MotorcycleAwarenessSystemTypes.hpp](#).

6.8.1.3 typedef int **DataPacket_t**[255]

Data packet for the V2V communication

Definition at line 47 of file [MotorcycleAwarenessSystemTypes.hpp](#).

6.8.1.4 typedef float **ObjectDistance**

Distance of an object from a detecting radar sensor.

Definition at line 31 of file [MotorcycleAwarenessSystemTypes.hpp](#).

6.8.1.5 typedef float **ObjectVelocity**

Velocity of an object detected by a radar sensor

Definition at line 32 of file [MotorcycleAwarenessSystemTypes.hpp](#).

6.8.2 Enumeration Type Documentation

6.8.2.1 enum **VehicleType**

Enum for vehicle type.

Enumerator

MOTORCYCLE
CAR

Definition at line 8 of file [MotorcycleAwarenessSystemTypes.hpp](#).

```
00009 {  
00010     MOTORCYCLE,  
00011     CAR  
00012 };
```

6.9 MotorcycleAwarenessSystemTypes.hpp

```
00001 #ifndef MOTORCYCLEAWARENESSSYSTEMTYPES_HPP  
00002 #define MOTORCYCLEAWARENESSSYSTEMTYPES_HPP  
00003  
00004 #include <list>  
00005 #include <map>  
00006  
00008 enum VehicleType  
00009 {
```

```
00010     MOTORCYCLE,
00011     CAR
00012 };
00013
00015 struct CanSignal_t
00016 {
00017     bool isBlinkerOn;
00018     int busData[16];
00019 };
00020
00021 typedef float Coordinate_t;
00022 typedef float currentTime_t;
00023 struct GpsSignal_t
00024 {
00025     Coordinate_t x;
00026     Coordinate_t y;
00027     currentTime_t currentTime;
00028 };
00029
00030
00031 typedef float ObjectDistance;
00032 typedef float ObjectVelocity;
00033 struct RadarSignal_t
00034 {
00035     ObjectDistance objectDistance;
00036     ObjectVelocity objectVelocity;
00037 };
00038
00039
00041 struct BlueToothMessage_t
00042 {
00043     bool isHazardPresent;
00044     unsigned int dataBuffer[255];
00045 };
00046
00047 typedef int DataPacket_t[255];
00048 struct MotorCycleLocation_t
00049 {
00050     DataPacket_t dataPacket;
00051 };
00052
00053 #endif
```

Index

- ~MotorcycleAwarenessSystem
 - MotorcycleAwarenessSystem, 8
- BluetoothMessage_t, 3
 - dataBuffer, 3
 - isHazardPresent, 3
- busData
 - CanSignal_t, 4
- CAR
 - MotorcycleAwarenessSystemTypes.hpp, 25
- canSignal
 - MotorcycleAwarenessSystem, 13
- CanSignal_t, 4
 - busData, 4
 - isBlinkerOn, 5
- carGpsSignal
 - MotorcycleAwarenessSystem, 13
- Coordinate_t
 - MotorcycleAwarenessSystemTypes.hpp, 25
- currentTime
 - GpsSignal_t, 6
- currentTime_t
 - MotorcycleAwarenessSystemTypes.hpp, 25
- dataBuffer
 - BluetoothMessage_t, 3
- dataPacket
 - MotorCycleLocation_t, 15
- DataPacket_t
 - MotorcycleAwarenessSystemTypes.hpp, 25
- GetMotorcycleLocation
 - MotorcycleAwarenessSystem, 9
- GpsSignal_t, 5
 - currentTime, 6
 - x, 6
 - y, 6
- Initialize
 - MotorcycleAwarenessSystem, 9
- isBlinkerOn
 - CanSignal_t, 5
- IsHazardPresent
 - MotorcycleAwarenessSystem, 9
- isHazardPresent
 - BluetoothMessage_t, 3
- IsMotorcycleInRange
 - MotorcycleAwarenessSystem, 10
- IsVehicleBlinkerOn
 - MotorcycleAwarenessSystem, 11
- MOTORCYCLE
 - MotorcycleAwarenessSystemTypes.hpp, 25
- main
 - main.cpp, 17
- main.cpp, 17
 - main, 17
- mainpage.dox, 19
- MonitorConditions
 - MotorcycleAwarenessSystem, 11
- MotorCycleLocation_t, 14
 - dataPacket, 15
- MotorcycleAwarenessSystemTypes.hpp
 - CAR, 25
 - MOTORCYCLE, 25
- MotorcycleAwarenessSystem, 6
 - ~MotorcycleAwarenessSystem, 8
 - canSignal, 13
 - carGpsSignal, 13
 - GetMotorcycleLocation, 9
 - Initialize, 9
 - IsHazardPresent, 9
 - IsMotorcycleInRange, 10
 - IsVehicleBlinkerOn, 11
 - MonitorConditions, 11
 - MotorcycleAwarenessSystem, 8
 - motorcycleGpsSignal, 13
 - motorcycleLocation, 14
 - motorcycleRadarSignal, 14
 - MotorcycleAwarenessSystem, 8
 - RelayWarningToOperator, 12
 - SAFETY_ZONE, 14
 - TrackMotorcycle, 13
 - vehicleType, 14
- MotorcycleAwarenessSystem.cpp, 19
- MotorcycleAwarenessSystem.hpp, 21
- MotorcycleAwarenessSystemTypes.hpp, 23
 - Coordinate_t, 25
 - currentTime_t, 25
 - DataPacket_t, 25
 - ObjectDistance, 25
 - ObjectVelocity, 25
 - VehicleType, 25
- motorcycleGpsSignal
 - MotorcycleAwarenessSystem, 13
- motorcycleLocation
 - MotorcycleAwarenessSystem, 14
- motorcycleRadarSignal
 - MotorcycleAwarenessSystem, 14
- ObjectDistance
 - MotorcycleAwarenessSystemTypes.hpp, 25
- objectDistance

- RadarSignal_t, [16](#)
- ObjectVelocity
 - MotorcycleAwarenessSystemTypes.hpp, [25](#)
- objectVelocity
 - RadarSignal_t, [16](#)
- RadarSignal_t, [15](#)
 - objectDistance, [16](#)
 - objectVelocity, [16](#)
- RelayWarningToOperator
 - MotorcycleAwarenessSystem, [12](#)
- SAFETY_ZONE
 - MotorcycleAwarenessSystem, [14](#)
- TrackMotorcycle
 - MotorcycleAwarenessSystem, [13](#)
- VehicleType
 - MotorcycleAwarenessSystemTypes.hpp, [25](#)
- vehicleType
 - MotorcycleAwarenessSystem, [14](#)
- x
 - GpsSignal_t, [6](#)
- y
 - GpsSignal_t, [6](#)