TecTask Oenay Can

Generated by Doxygen 1.8.14

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 AudiVehicle Class Reference

`#include <AudiVehicle.h>`

Public Member Functions

- AudiVehicle ()
- AudiVehicle (string _type)
- void StartEngine (vector< double > direction, double _velocity, double _duration)
- void ClockTime_and_UpdatePosition (double _duration)
- vector< Protocols * > get_protocols ()
- virtual ~AudiVehicle ()

Private Attributes

- string type
- double length
- double width
- vector< double * > position
- vector< double > direction
- double * drivetime
- double velocity
- vector< ECU * > ecus
- vector< Sensors * > sensors
- vector< Protocols * > protocols

4.1.1 Detailed Description

Class AudiVehicle contains a single generic vehicle where the actors, sensors, agents protocolls and controll units to be attached. Mechanics and dimensions as well as the kinematics of the vehicle is filled by means of the constructers and other public functions

Definition at line 10 of file AudiVehicle.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AudiVehicle() [1/2]

```
AudiVehicle::AudiVehicle ( )
```

The default constructer of this principally doing nothing but providing new assignment which is used in this context.

Definition at line 24 of file AudiVehicle.cpp.

4.1.2.2 AudiVehicle() [2/2]

```
AudiVehicle::AudiVehicle (
            string _type )
```

This constructer prepares the vehicle before the run. Based on a car pool (enumurator), the constructer sets the relatinships in between sensors and protocols. ECU's and more are also declared in this content.

Definition at line 28 of file AudiVehicle.cpp.

4.1.2.3 ~AudiVehicle()

```
AudiVehicle::~AudiVehicle ( )  [virtual]
```

Definition at line 130 of file AudiVehicle.cpp.

4.1.3 Member Function Documentation

4.1.3.1 ClockTime_and_UpdatePosition()

```
void AudiVehicle::ClockTime_and_UpdatePosition (
            double _duration )
```

A function which is threaded. This function can be considered as a wheel or GPS sensor. Better would be to inherit another sensor from the parent and read those variables accordingly. But since any actor, including the valves, brakes and especiall engine is missing in this context, a simple thread is replaced this purpose.

Definition at line 115 of file AudiVehicle.cpp.

4.1.3.2 get_protocols()

```
vector<Protocols*> AudiVehicle::get_protocols ( )  [inline]
```

Definition at line 70 of file AudiVehicle.h.

4.1.3.3 StartEngine()

```
void AudiVehicle::StartEngine (
            vector< double > direction,
            double _velocity,
            double _duration )
```

This function does not necessarily defines the idle condition. It is the enterance to the real time environment, thus the kinematics (velcity) is in this case quite crucial to determine the behavior of the threads.

Definition at line 95 of file AudiVehicle.cpp.

4.1.4 Member Data Documentation

4.1.4.1 direction

```
vector<double> AudiVehicle::direction  [private]
```

Definition at line 27 of file AudiVehicle.h.

4.1.4.2 drivetime

```
double* AudiVehicle::drivetime  [private]
```

The driver time is representing the real time.

Definition at line 31 of file AudiVehicle.h.

4.1.4.3 ecus

```
vector<ECU*> AudiVehicle::ecus  [private]
```

The vehicle can contain more then one ECU. The relationships in between protocolls and sensors are contained in this vector of pointers.

Definition at line 39 of file AudiVehicle.h.

4.1.4.4 length

```
double AudiVehicle::length  [private]
```

The physical duimensions of the vehicle. Those are used to place the sensors.

Definition at line 21 of file AudiVehicle.h.

4.1.4.5 position

```
vector<double*> AudiVehicle::position  [private]
```

The position pointer vector as well as the direction. Especially the position will be threaded in this code.

Definition at line 26 of file AudiVehicle.h.

4.1.4.6 protocols

```
vector<Protocols*> AudiVehicle::protocols  [private]
```

Many protocols, for example CAN bus and other fast ethernets can be implemented in this model.

Definition at line 47 of file AudiVehicle.h.

4.1.4.7 sensors

```
vector<Sensors*> AudiVehicle::sensors  [private]
```

The vehicle can again contain many sensors.

Definition at line 43 of file AudiVehicle.h.

4.1.4.8 type

```
string AudiVehicle::type  [private]
```

The type is defining the type of the vehicle as string. The configuration can be extended by means of the eneumerator.

Definition at line 17 of file AudiVehicle.h.

4.1.4.9 velocity

```
double AudiVehicle::velocity  [private]
```

The velocity is considered to be an input to the vehicle class.

Definition at line 35 of file AudiVehicle.h.

4.1.4.10 width

```
double AudiVehicle::width  [private]
```

Definition at line 22 of file AudiVehicle.h.

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

- TecTask_Oenay_Can/AudiVehicle.h
- TecTask_Oenay_Can/AudiVehicle.cpp

4.2 DriverAssistance Class Reference

```
#include <DriverAssistance.h>
```

Public Member Functions

- DriverAssistance ()
- void launch (vector< Sensors *> _AudiVehicleSensors, vector< double *> _position, double *_drivetime, double _duration)
- void position_log_thread (vector< double *> _position, double *_drivetime, double _duration)
- void clock_thread_function ()
- int SensorPreFusion (vector< Sensors *> _AudiVehicleSensors, double *_drivetime, double _duration)
- virtual ~DriverAssistance ()

Public Attributes

- ofstream positionlog
- ofstream sensorprefusionlog

Private Attributes

- double * timestamp

4.2.1 Detailed Description

Class Driverassistance is almost public. Only the time stamp is considered to be public. The core component of the task, namely sensorprefusion is implemented in this content.

Definition at line 19 of file DriverAssistance.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 DriverAssistance()

```
DriverAssistance::DriverAssistance ( )
```

The default constructer

Definition at line 7 of file DriverAssistance.cpp.

4.2.2.2 ~DriverAssistance()

```
DriverAssistance::~DriverAssistance ( )  [virtual]
```

Definition at line 114 of file DriverAssistance.cpp.

4.2.3 Member Function Documentation

4.2.3.1 clock_thread_function()

```
void DriverAssistance::clock_thread_function ( )
```

This function can be used to consider the time elapse for communication.

Definition at line 19 of file DriverAssistance.cpp.

4.2.3.2 launch()

```
void DriverAssistance::launch (
            vector< Sensors *> _AudiVehicleSensors,
            vector< double *> _position,
            double * _drivetime,
            double _duration )
```

The launcher of the driving assistance is simply one of the core locations of the task.

Definition at line 72 of file DriverAssistance.cpp.

4.2.3.3 position_log_thread()

```
void DriverAssistance::position_log_thread (
            vector< double *> _position,
            double * _drivetime,
            double _duration )
```

A function which is used as thread to print and update global vehicle position and the real time.

Definition at line 12 of file DriverAssistance.cpp.

4.2.3.4 SensorPreFusion()

```
int DriverAssistance::SensorPreFusion (
            vector< Sensors *> _AudiVehicleSensors,
            double * _drivetime,
            double _duration )
```

Sensor pre fusion listens all the individual threads of the sensors and determines catches based on the given criteria.

Definition at line 28 of file DriverAssistance.cpp.

4.2.4 Member Data Documentation

4.2.4.1 positionlog

```
ofstream DriverAssistance::positionlog
```

A log file for the thread of the position and the driver time. The data types are precise. However the required datatypes for sensor are implemented in the corresponding places

Definition at line 32 of file DriverAssistance.h.

4.2.4.2 sensorprefusionlog

```
ofstream DriverAssistance::sensorprefusionlog
```

The prefusion log file for the sensors. Here, only the timestamp and the catches are printed

Definition at line 36 of file DriverAssistance.h.

4.2.4.3 timestamp

```
double* DriverAssistance::timestamp  [private]
```

Definition at line 22 of file DriverAssistance.h.

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

- TecTask_Oenay_Can/DriverAssistance.h
- TecTask_Oenay_Can/DriverAssistance.cpp

4.3 ECU Class Reference

```
#include <ECU.h>
```

Public Member Functions

- ECU ()
- void SetPhysicalConnections (int _protocolid, int _sensorid)
- void run (vector< double *> _position, double *_drivetime, double _duration)
- virtual ~ECU ()

Public Attributes

- vector< Sensors * > AudiVehicleSensors
- vector< Protocols * > AudiVehicleProtocols

Private Attributes

- map< int, int > sense2recieve2this
- DriverAssistance RunTimeEnvironment

4.3.1 Detailed Description

Class ECU contains mainly two privat members. The first one defined the hashtag from sensors to protocolls, but this is not used in this content. The idea was to simulate the time gap based on the communication elapse. For this problem the ethernet listener and emitter is assummed to be fast enough. The second private member is the runtimeenvironment as a driver assistance system which does not activate any actors and does not take action but listens carefully.

Definition at line 22 of file ECU.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ECU()

```
ECU::ECU ( )
```

The constructer

Definition at line 5 of file ECU.cpp.

4.3.2.2 ~ECU()

```
ECU::~ECU ( )  [virtual]
```

Definition at line 19 of file ECU.cpp.

4.3.3 Member Function Documentation

4.3.3.1 run()

```
void ECU::run (
            vector< double *> _position,
            double * _drivetime,
            double _duration )
```

The run calls the runtime environment. A gateway to the core of the problem.

Definition at line 10 of file ECU.cpp.

4.3.3.2 SetPhysicalConnections()

```
void ECU::SetPhysicalConnections (
            int _protocolid,
            int _sensorid )
```

This function inserts members to the hashtag (private map)

Definition at line 15 of file ECU.cpp.

4.3.4 Member Data Documentation

4.3.4.1 AudiVehicleProtocols

```
vector<Protocols*> ECU::AudiVehicleProtocols
```

The vehicle hardware busses are considered to be public members of ECU, since they do not really belong to ECU but to the vehicle. ECU is suppossed to communicate with those.

Definition at line 39 of file ECU.h.

4.3.4.2 AudiVehicleSensors

```
vector<Sensors*> ECU::AudiVehicleSensors
```

The vehicle sensors are considered to be public members of ECU, since they do not really belong to ECU but to the vehicle. ECU is suppossed to communicate with those.

Definition at line 35 of file ECU.h.

4.3.4.3 RunTimeEnvironment

```
DriverAssistance ECU::RunTimeEnvironment  [private]
```

Definition at line 26 of file ECU.h.

4.3.4.4 sense2recieve2this

```
map<int, int> ECU::sense2recieve2this  [private]
```

Definition at line 25 of file ECU.h.

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

- TecTask_Oenay_Can/ECU.h
- TecTask_Oenay_Can/ECU.cpp

4.4 Environment Class Reference

```
#include <ObjEnvironment.h>
```

Public Member Functions

- Environment (string filename)
- ~Environment ()

Public Attributes

- vector< vector< double > > real_object_coordinates
- vector< int > real_object_type

4.4.1 Detailed Description

For test cases, the real environment is simulated via an input file. This file can contain many lines to detect many objects, but the thread structres are not configured for this! I would not recommend to do it. The first three numbers are the coordinates of the object (not relative), the third one is the objectclass.

Definition at line 17 of file ObjEnvironment.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Environment()

```
Environment::Environment (
            string filename )
```

Definition at line 5 of file ObjEnvironment.cpp.

4.4.2.2 ~Environment()

```
Environment::~Environment ( )
```

Definition at line 30 of file ObjEnvironment.cpp.

4.4.3 Member Data Documentation

4.4.3.1 real_object_coordinates

```
vector<vector<double> > Environment::real_object_coordinates
```

Definition at line 20 of file ObjEnvironment.h.

4.4.3.2 real_object_type

```
vector<int> Environment::real_object_type
```

Definition at line 21 of file ObjEnvironment.h.

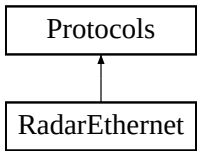
The documentation for this class was generated from the following files:

- TecTask_Oenay_Can/ObjEnvironment.h
- TecTask_Oenay_Can/ObjEnvironment.cpp

4.5 Protocols Class Reference

```
#include <Protocols.h>
```

Inheritance diagram for Protocols:



Public Member Functions

- Protocols ()
- Protocols (int _id, bool _on_off=false, string _type="None")
- virtual bool get_status ()
- virtual string get_type ()
- virtual int get_id ()
- virtual int get_bits_per_ms ()
- virtual void print_this ()
- virtual ~Protocols ()

Protected Attributes

- int id
- bool on_off
- string type

4.5.1 Detailed Description

Class protocolls represent a polymorphic inheritance of the hardware protocolls. Both the parent and the child classes are constructed but later on not really used. For an extension problem those classes might be usefull.

Definition at line 10 of file Protocols.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Protocols() [1/2]

```
Protocols::Protocols ( )
```

Definition at line 8 of file Protocols.cpp.

4.5.2.2 Protocols() [2/2]

```
Protocols::Protocols (
            int _id,
            bool _on_off = false,
            string _type = "None" )
```

Definition at line 12 of file Protocols.cpp.

4.5.2.3 ~Protocols()

```
Protocols::~Protocols ( )  [virtual]
```

Definition at line 39 of file Protocols.cpp.

4.5.3 Member Function Documentation

4.5.3.1 get_bits_per_ms()

```
virtual int Protocols::get_bits_per_ms ( )  [inline], [virtual]
```

Reimplemented in RadarEthernet.

Definition at line 22 of file Protocols.h.

4.5.3.2 get_id()

```
int Protocols::get_id ( )  [virtual]
```

Reimplemented in RadarEthernet.

Definition at line 29 of file Protocols.cpp.

4.5.3.3 get_status()

```
bool Protocols::get_status ( )  [virtual]
```

Definition at line 19 of file Protocols.cpp.

4.5.3.4 get_type()

```
string Protocols::get_type ( )  [virtual]
```

Definition at line 24 of file Protocols.cpp.

4.5.3.5 print_this()

```
void Protocols::print_this ( )  [virtual]
```

Reimplemented in RadarEthernet.

Definition at line 33 of file Protocols.cpp.

4.5.4 Member Data Documentation

4.5.4.1 id

```
int Protocols::id  [protected]
```

Definition at line 13 of file Protocols.h.

4.5.4.2 on_off

```
bool Protocols::on_off  [protected]
```

Definition at line 14 of file Protocols.h.

4.5.4.3 type

```
string Protocols::type  [protected]
```

Definition at line 15 of file Protocols.h.

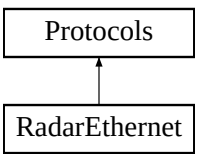
The documentation for this class was generated from the following files:

- TecTask_Oenay_Can/Protocols.h
- TecTask_Oenay_Can/Protocols.cpp

4.6 RadarEthernet Class Reference

```
#include <Protocols.h>
```

Inheritance diagram for RadarEthernet:



Public Member Functions

- RadarEthernet (int _id, bool _onoff=false, int _bits_per_ms=0)
- virtual int get_bits_per_ms ()
- virtual int get_id ()
- virtual void print_this ()

Private Attributes

- int bits_per_ms

Additional Inherited Members

4.6.1 Detailed Description

Definition at line 27 of file Protocols.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 RadarEthernet()

```
RadarEthernet::RadarEthernet (
            int _id,
            bool _onoff = false,
            int _bits_per_ms = 0 )
```

Definition at line 43 of file Protocols.cpp.

4.6.3 Member Function Documentation

4.6.3.1 get_bits_per_ms()

```
virtual int RadarEthernet::get_bits_per_ms ( )  [inline], [virtual]
```

Reimplemented from Protocols.

Definition at line 32 of file Protocols.h.

4.6.3.2 get_id()

```
virtual int RadarEthernet::get_id ( )  [inline], [virtual]
```

Reimplemented from Protocols.

Definition at line 33 of file Protocols.h.

4.6.3.3 print_this()

```
void RadarEthernet::print_this ( )  [virtual]
```

Reimplemented from Protocols.

Definition at line 45 of file Protocols.cpp.

4.6.4 Member Data Documentation

4.6.4.1 bits_per_ms

```
int RadarEthernet::bits_per_ms  [private]
```

Definition at line 29 of file Protocols.h.

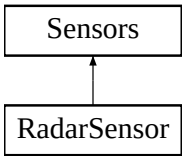
The documentation for this class was generated from the following files:

- TecTask_Oenay_Can/Protocols.h
- TecTask_Oenay_Can/Protocols.cpp

4.7 RadarSensor Class Reference

```
#include <Sensors.h>
```

Inheritance diagram for RadarSensor:



Public Member Functions

- RadarSensor (int _id, vector< double > _pos, vector< double > _range, string _envfile, bool _onoff=false, type_timestamp _timestamp=0, type_objectclass _objectClass=0)
- virtual int get_id ()
- virtual void SenseObjEnvironment (vector< double *> _position, double *_drivetime, double _duration)
- virtual type_timestamp Get_timestamp ()
- virtual type_objectclass Get_objectClass ()
- virtual vector< type_unscaled_obj_coords > Get_objectCoords ()
- virtual void print_this ()

Public Attributes

- ofstream catchlog

Private Attributes

- vector< type_unscaled_obj_coords > obj_coords
- type_timestamp timestamp
- type_objectclass objectClass
- vector< double > range
- vector< double > r_pos

Additional Inherited Members

4.7.1 Detailed Description

Definition at line 46 of file Sensors.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 RadarSensor()

```
RadarSensor::RadarSensor (
            int _id,
            vector< double > _pos,
            vector< double > _range,
            string _envfile,
            bool _onoff = false,
            type_timestamp _timestamp = 0,
            type_objectclass _objectClass = 0 )
```

Definition at line 55 of file Sensors.cpp.

4.7.3 Member Function Documentation

4.7.3.1 get_id()

```
virtual int RadarSensor::get_id ( )  [inline], [virtual]
```

Reimplemented from Sensors.

Definition at line 56 of file Sensors.h.

4.7.3.2 Get_objectClass()

```
virtual type_objectclass RadarSensor::Get_objectClass ( )  [inline], [virtual]
```

The typedef objectClass (as required in the question) shall be delivered normally to the run time environment via ethernet, but get is used for this model.

Reimplemented from Sensors.

Definition at line 69 of file Sensors.h.

4.7.3.3 Get_objectCoords()

```
virtual vector<type_unscaled_obj_coords> RadarSensor::Get_objectCoords ( )  [inline], [virtual]
```

The typedef unscaled (as well as scaled) coordinates (as required in the question) shall be delivered normally to the run time environment via ethernet, but get is used for this model.

Reimplemented from Sensors.

Definition at line 73 of file Sensors.h.

4.7.3.4 Get_timestamp()

```
virtual type_timestamp RadarSensor::Get_timestamp ( )  [inline], [virtual]
```

The typedef timestamp (as required in the question) shall be delivered normally to the run time environment via ethernet, but get is used for this model.

Reimplemented from Sensors.

Definition at line 65 of file Sensors.h.

4.7.3.5 print_this()

```
void RadarSensor::print_this ( )  [virtual]
```

Reimplemented from Sensors.

Definition at line 70 of file Sensors.cpp.

4.7.3.6 SenseObjEnvironment()

```
void RadarSensor::SenseObjEnvironment (
            vector< double *> _position,
            double * _drivetime,
            double _duration )  [virtual]
```

The corresponding virtual twin in the parent class is obsolete.

- This function connects the sensor via the environment class. The thread decides if the sensor can emit signals from the object or not.

SCALING!

Reimplemented from Sensors.

Definition at line 78 of file Sensors.cpp.

4.7.4 Member Data Documentation

4.7.4.1 catchlog

```
ofstream RadarSensor::catchlog
```

Definition at line 55 of file Sensors.h.

4.7.4.2 obj_coords

```
vector<type_unscaled_obj_coords> RadarSensor::obj_coords  [private]
```

Definition at line 48 of file Sensors.h.

4.7.4.3 objectClass

```
type_objectclass RadarSensor::objectClass  [private]
```

Definition at line 50 of file Sensors.h.

4.7.4.4 r_pos

```
vector<double> RadarSensor::r_pos  [private]
```

Definition at line 52 of file Sensors.h.

4.7.4.5 range

```
vector<double> RadarSensor::range  [private]
```

Definition at line 51 of file Sensors.h.

4.7.4.6 timestamp

```
type_timestamp RadarSensor::timestamp  [private]
```

Definition at line 49 of file Sensors.h.

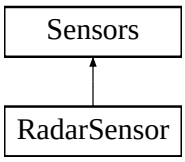
The documentation for this class was generated from the following files:

- TecTask_Oenay_Can/Sensors.h
- TecTask_Oenay_Can/Sensors.cpp

4.8 Sensors Class Reference

```
#include <Sensors.h>
```

Inheritance diagram for Sensors:



Public Member Functions

- Sensors ()
- Sensors (int _id, vector< double > _pos, string _envfile, bool initial_off=false, string _type="None")
- virtual bool get_status ()
- virtual string get_type ()
- virtual int get_id ()
- virtual void print_this ()
- virtual void SenseObjEnvironment (vector< double *> _position, double *_drivetime, double _duration)
- virtual type_timestamp Get_timestamp ()
- virtual type_objectclass Get_objectClass ()
- virtual vector< type_unscaled_obj_coords > Get_objectCoords ()
- bool operator< (const Sensors &rhs) const
- virtual ~Sensors ()

Protected Attributes

- int id
- bool on_off
- string type
- vector< double > pos
- Environment * env

4.8.1 Detailed Description

Definition at line 23 of file Sensors.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Sensors() [1/2]

```
Sensors::Sensors ( )
```

Definition at line 13 of file Sensors.cpp.

4.8.2.2 Sensors() [2/2]

```
Sensors::Sensors (
            int _id,
            vector< double > _pos,
            string _envfile,
            bool initial_off = false,
            string _type = "None" )
```

Definition at line 17 of file Sensors.cpp.

4.8.2.3 ~Sensors()

```
Sensors::~Sensors ( )  [virtual]
```

Definition at line 51 of file Sensors.cpp.

4.8.3 Member Function Documentation

4.8.3.1 get_id()

```
int Sensors::get_id ( )  [virtual]
```

Reimplemented in RadarSensor.

Definition at line 39 of file Sensors.cpp.

4.8.3.2 Get_objectClass()

```
virtual type_objectclass Sensors::Get_objectClass ( )  [inline], [virtual]
```

Reimplemented in RadarSensor.

Definition at line 40 of file Sensors.h.

4.8.3.3 Get_objectCoords()

```
virtual vector<type_unscaled_obj_coords> Sensors::Get_objectCoords ( )  [inline], [virtual]
```

Reimplemented in RadarSensor.

Definition at line 41 of file Sensors.h.

4.8.3.4 get_status()

```
bool Sensors::get_status ( )  [virtual]
```

Definition at line 29 of file Sensors.cpp.

4.8.3.5 Get_timestamp()

```
virtual type_timestamp Sensors::Get_timestamp ( )  [inline], [virtual]
```

Reimplemented in RadarSensor.

Definition at line 39 of file Sensors.h.

4.8.3.6 get_type()

```
string Sensors::get_type ( )  [virtual]
```

Definition at line 34 of file Sensors.cpp.

4.8.3.7 operator<()

```
bool Sensors::operator< (
            const Sensors & rhs ) const  [inline]
```

Definition at line 42 of file Sensors.h.

4.8.3.8 print_this()

```
void Sensors::print_this ( )  [virtual]
```

Reimplemented in RadarSensor.

Definition at line 45 of file Sensors.cpp.

4.8.3.9 SenseObjEnvironment()

```
virtual void Sensors::SenseObjEnvironment (
            vector< double *> _position,
            double * _drivetime,
            double _duration )  [inline], [virtual]
```

Reimplemented in RadarSensor.

Definition at line 38 of file Sensors.h.

4.8.4 Member Data Documentation

4.8.4.1 env

```
Environment* Sensors::env  [protected]
```

Definition at line 30 of file Sensors.h.

4.8.4.2 id

```
int Sensors::id  [protected]
```

Definition at line 26 of file Sensors.h.

4.8.4.3 on_off

```
bool Sensors::on_off  [protected]
```

Definition at line 27 of file Sensors.h.

4.8.4.4 pos

```
vector<double> Sensors::pos  [protected]
```

Definition at line 29 of file Sensors.h.

4.8.4.5 type

```
string Sensors::type  [protected]
```

Definition at line 28 of file Sensors.h.

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

- TecTask_Oenay_Can/Sensors.h
- TecTask_Oenay_Can/Sensors.cpp

Chapter 5

File Documentation

5.1 TecTask_Oenay_Can/AudiVehicle.cpp File Reference

```
#include "AudiVehicle.h"
```

Enumerations

- enum CarPool { ExampleConfigCar, AnotherTestCar, SomeExpensiveCar, InvalidOption }

Functions

- CarPool CarofConcern (string _type)

5.1.1 Enumeration Type Documentation

5.1.1.1 CarPool

```
enum CarPool
```

Enumerator

ExampleConfigCar	
AnotherTestCar	
SomeExpensiveCar	
InvalidOption	

Definition at line 4 of file AudiVehicle.cpp.

5.1.2 Function Documentation

5.1.2.1 CarofConcern()

```
CarPool CarofConcern (
            string _type )
```

Definition at line 11 of file AudiVehicle.cpp.

5.2 AudiVehicle.cpp

```
00001 #include "AudiVehicle.h"
00002
00003
00004 enum CarPool {
00005     ExampleConfigCar,
00006     AnotherTestCar,
00007     SomeExpensiveCar,
00008     InvalidOption
00009 };
00010
00011 CarPool CarofConcern(string _type) {
00012     map<string, CarPool> type2enum;
00013     type2enum["ExampleConfigCar"] = ExampleConfigCar;
00014     type2enum["AnotherTestCar"] = AnotherTestCar;
00015     type2enum["SomeExpensiveCar"] = SomeExpensiveCar;
00016     map<string, CarPool>::iterator it;
00017     it = type2enum.find(_type);
00018     if (it != type2enum.end())
00019         return it->second;
00020     return InvalidOption;
00021
00022 }
00023
00024 AudiVehicle::AudiVehicle()
00025 {
00026 }
00027
00028 AudiVehicle::AudiVehicle(string _type) {
00029
00030     switch (CarofConcern(_type)) {
00031         case ExampleConfigCar: {
00032             cout <<"You have selected the car type from the car pool: "<<_type << endl;
00033             type = _type;
00034             length = 5.172;
00035             width = 1.945;
00036             velocity = 0.0;
00037             for (int i = 0; i < 3; i++)
00038             {
00039                 double* zero = new double(0.0);
00040                 position.push_back(zero);
00041                 direction.push_back(0.0);
00042             }
00043
00044             //Begin: Decleration of ECU's
00045             ECU* ecuofthis = new ECU();
00046             ecus.push_back(ecuofthis);
00047             //End: Decleration of ECU's
00048
00049
00050             //Begin: Decleration and definition of Sensors
00051             for (int s = 0; s < 4; s++)
00052             {
00053                 vector<double> current_pos;
00054                 vector<double> range;
00055                 range.push_back(3.0);
00056                 range.push_back(3.0);
00057                 current_pos.push_back(s / 2 - width / 2.0);
00058                 current_pos.push_back(s % 2 - length / 2.0);
00059                 current_pos.push_back(0.3);
00060
00061                 RadarSensor* sensorofthis = new RadarSensor(s,current_pos,range,"
```

```
      Environment.env",false, 0);
00062                 sensors.push_back(sensorofthis);
00063                 for (unsigned int e = 0; e < ecus.size(); e++)
00064                     ecus[e]->AudiVehicleSensors.push_back(sensorofthis);
00065             }
00066             for (unsigned int i = 0; i < sensors.size(); i++)
00067                 sensors[i]->print_this();
00068             //End: Decleration and definition of Sensors
00069
00070             //Begin: Decleration and definition of Hardware Protocols
00071             RadarEthernet* ethernetofthis = new RadarEthernet(0,false,1000);
00072             protocols.push_back(ethernetofthis);
00073             for (unsigned int e = 0; e < ecus.size(); e++)
00074                 ecus[e]->AudiVehicleProtocols.push_back(ethernetofthis);
00075             for (unsigned int i = 0; i < protocols.size(); i++)
00076                 protocols[i]->print_this();
00077
00078             //End: Decleration and definition of Hardware Protocols
00079
00080             //Begin: Definition of the ECU Physical Protocol hashtags
00081             for (unsigned int e = 0; e < ecus.size(); e++)
00082                 for (unsigned int i = 0; i < sensors.size(); i++)
00083                     ecus[e]->SetPhysicalConnections(protocols[0]->get_id(),
      sensors[i]->get_id());
00084             //End: Definition of the ECU Physical Protocol hashtags
00085             break;
00086         }
00087         case AnotherTestCar: {cout << "Not implemented Yet" << endl; break; }
00088         case SomeExpensiveCar: {cout << "Not implemented Yet" << endl; break; }
00089         case InvalidOption: {cout << "The car string you have entered could not be mapped to
       the enumerator" << endl; break; }
00090         default: {cout<<"one should never see this command output"<<endl; }
00091
00092     }
00093 }
00094
00095 void AudiVehicle::StartEngine(vector<double> _direction, double _velocity, double
      _duration)
00096 {
00097     cout << "The Engine has started with initial velocity and position. Launching the ECU's." << endl;
00098     direction = _direction;
00099     velocity = _velocity;
00100
00101     thread position_thread(&AudiVehicle::ClockTime_and_UpdatePosition
      ,this, _duration);
00102
00103
00104     for (unsigned int e = 0; e < ecus.size(); e++)
00105         ecus[e]->run(position,drivetime,_duration);
00106
00107     if (position_thread.joinable())
00108     {
00109         cout << "Joining Position Thread " << std::endl;
00110         position_thread.join();
00111     }
00112
00113 }
00114
00115 void AudiVehicle::ClockTime_and_UpdatePosition(double _duration)
00116 {
00117     drivetime = new double(0.0);
00118     while (*(this->drivetime) < _duration) {
00119         auto t1 = chrono::high_resolution_clock::now();
00120         //sleeping 1 milliseconds
00121         Sleep(1);
00122         auto t2 = chrono::high_resolution_clock::now();
00123         chrono::duration<double, milli> time_elapsed = t2 - t1;
00124         *(this->drivetime) = *(this->drivetime) + time_elapsed.count();
00125         for (int i = 0; i < 3; i++)
00126             *(this->position[i]) = *(this->position[i]) +
      velocity * direction[i] * time_elapsed.count();
00127     }
00128 }
00129
00130 AudiVehicle::~AudiVehicle()
00131 {
00132
00133 }
```

5.3 TecTask_Oenay_Can/AudiVehicle.h File Reference

```
#include "ECU.h"
```

```
#include "miscellaneous.h"
```

Classes

- class AudiVehicle

5.3.1 Detailed Description

Contains only the Vehicle Class.

Definition in file AudiVehicle.h.

5.4 AudiVehicle.h

```
00001 #pragma once
00002 #include "ECU.h"
00003 #include "miscellaneous.h"
00004
00010 class AudiVehicle
00011 {
00012 private:
00017     string type;
00021     double length;
00022     double width;
00026     vector<double*> position;
00027     vector<double> direction;
00031     double* drivetime;
00035     double velocity;
00039     vector<ECU*> ecus;
00043     vector<Sensors*> sensors;
00047     vector<Protocols*> protocols;
00048
00049
00050 public:
00054     AudiVehicle();
00059     AudiVehicle(string _type);
00064     void StartEngine(vector<double> direction, double _velocity, double _duration);
00069     void ClockTime_and_UpdatePosition(double _duration);
00070     vector<Protocols*> get_protocols() { return protocols;  };
00071     virtual ~AudiVehicle();
00072 };
00073
```

5.5 TecTask_Oenay_Can/Driver.cpp File Reference

```
#include <iostream>
#include <thread>
#include "Header.h"
```

Functions

- int main ()

5.5.1 Detailed Description

Contains only the main function

Definition in file Driver.cpp.

5.5.2 Function Documentation

5.5.2.1 main()

```
int main ( )
```

in m/ms

Definition at line 15 of file Driver.cpp.

5.6 Driver.cpp

```
00001 // test1.cpp: Definiert den Einstiegspunkt für die Konsolenanwendung.
00002 //
00003
00004 #include <iostream>
00005
00006 #include <thread>
00007 #include "Header.h"
00008
00009 using namespace std;
00010
00015 int main()
00016 {
00017
00018
00019     cout << "Selecting the Car" << endl;
00020     AudiVehicle HisVehicle("ExampleConfigCar");
00021     vector<double> direction(3);
00022     direction[0] = 0.0;
00023     direction[1] = 1.0;
00024     direction[2] = 0.0;
00025     double velocity = 0.00972222;
00026     double duration(2500.0);
00027
00028     HisVehicle.StartEngine(direction,velocity,duration);
00029
00030     return 0;
00031 }
00032
```

5.7 TecTask_Oenay_Can/DriverAssistance.cpp File Reference

```
#include "DriverAssistance.h"
#include <iostream>
#include "miscellaneous.h"
```

5.8 DriverAssistance.cpp

```
00001 #include "DriverAssistance.h"
00002 #include <iostream>
00003 #include"miscellaneous.h"
00004 using namespace std;
00005
00006
00007 DriverAssistance::DriverAssistance()
00008 {
00009
00010 }
00011
00012 void DriverAssistance::position_log_thread(vector<double*> _position,
      double* _drivetime, double _duration) {
00013
00014     while (*_drivetime<_duration) {
00015         this->positionlog << *_drivetime <<" "<<*_position[0] <<" "<< *_position[1] <<" "<< *_position[2]
      <<endl;
00016     }
00017     }
00018
00019 void DriverAssistance::clock_thread_function() {
00020     auto tstart = chrono::high_resolution_clock::now();
00021     while (true) {
00022         auto tend = chrono::high_resolution_clock::now();
00023         chrono::duration<double, std::milli> time_elapsed = tend - tstart;
00024         *(this->timestamp) = time_elapsed.count();
00025     }
00026 }
00027
00028 int DriverAssistance::SensorPreFusion(vector<Sensors*> _AudiVehicleSensors
      , double* _drivetime, double _duration) {
00029     unsigned short int catches = 0;
00030     vector<unsigned short int> catchers;
00031     while (*_drivetime < _duration) {
00032         type_objectclass catched = 0;
00033         for (unsigned int i = 0; i < _AudiVehicleSensors.size(); i++)
00034         {
00035             if (Ltwnorm(_AudiVehicleSensors[i]->Get_objectCoords())!=0)
00036             for (unsigned int j = i+1; j < _AudiVehicleSensors.size(); j++)
00037             {
00038                 if (_AudiVehicleSensors[i]->Get_objectClass() == _AudiVehicleSensors[j]->Get_objectClass())
00039                 {
00040                     if (Ltwnorm(_AudiVehicleSensors[i]->Get_objectCoords()) != 0)
00041                         if(_AudiVehicleSensors[i]->Get_objectCoords()[0]== _AudiVehicleSensors[j]->
      Get_objectCoords()[0])
00042                             if (_AudiVehicleSensors[i]->Get_objectCoords()[1] == _AudiVehicleSensors[j]->
      Get_objectCoords()[1])
00043                                 if (_AudiVehicleSensors[i]->Get_objectCoords()[2] == _AudiVehicleSensors[j]
      ->Get_objectCoords()[2])
00044                                 {
00045                                     catched = _AudiVehicleSensors[j]->Get_objectClass();
00046                                     catchers.push_back(i);
00047                                     catchers.push_back(j);
00048                                     catches++;
00049                                     goto there;
00050                                 }
00051                 }
00052             }
00053         }
00054         there:
00055         {
00056             if (catches == 2)
00057             {
00058                 catches--;
00059                 cout <<"At real time: "<<*_drivetime<<" the objectclass "<<unsigned(catched)
00060                     <<" is detected at least by sensors "<<to_string(catchers[0])<<" and "<< to_string(
      catchers[1]) <<endl;
00061
00062
00063             }
00064         }
00065         sensorprefusionlog << *_drivetime << " " << unsigned(catched) << endl;
00066
00067         Sleep(10);
00068     }
00069     return 0;
00070 }
00071
00072 void DriverAssistance::launch(vector<Sensors*> _AudiVehicleSensors, vector<double*>
       _position, double* _drivetime, double _duration){
00073
00074     vector<thread> SensorRecieverThreads;
00075     timestamp = new double(0.0);
00076
```

```
00077     positionlog.open("Threadlog_Wheel_and_GPS_position.log", ofstream::out);
00078     thread position_log_thread(&DriverAssistance::position_log_thread,
      this,_position,_drivetime, _duration);
00079
00080     for (unsigned int i = 0; i < _AudiVehicleSensors.size(); i++)
00081     {
00082         thread SenseObjEnvThread(&Sensors::SenseObjEnvironment,
      _AudiVehicleSensors[i], _position, _drivetime, _duration);
00083         SensorRecieverThreads.push_back(move(SenseObjEnvThread));
00084     }
00085
00086     sensorprefusionlog.open("SensorPreFusion.log", ofstream::out);
00087     thread sensorprefusionthread(&DriverAssistance::SensorPreFusion, this,
       _AudiVehicleSensors, _drivetime, _duration);
00088
00089
00090     if (position_log_thread.joinable())
00091     {
00092         cout << "Joining Position log Thread " << std::endl;
00093         position_log_thread.join();
00094     }
00095     for (unsigned int i = 0; i < SensorRecieverThreads.size(); i++)
00096     if (SensorRecieverThreads[i].joinable())
00097     {
00098         cout << "Joining sense environment Thread " <<to_string(_AudiVehicleSensors[i]->get_id())<<
      std::endl;
00099         SensorRecieverThreads[i].join();
00100     }
00101
00102     if (sensorprefusionthread.joinable())
00103     {
00104         cout << "Joining Sensor Pre Fusion Thread " << std::endl;
00105         sensorprefusionthread.join();
00106     }
00107
00108     //cout << SensorPreFusion() << endl;
00109
00110
00111 }
00112
00113
00114 DriverAssistance::~DriverAssistance()
00115 {
00116 }
```

5.9 TecTask_Oenay_Can/DriverAssistance.h File Reference

```
#include <iostream>
#include <vector>
#include <map>
#include "Protocols.h"
#include "Sensors.h"
#include <Windows.h>
#include <thread>
#include <fstream>
#include <chrono>
```

Classes

- class DriverAssistance

5.9.1 Detailed Description

Contains only the DriverAssistance class and the real time environment variable of that type

Definition in file DriverAssistance.h.

5.10 DriverAssistance.h

```
00001 #pragma once
00002 #include <iostream>
00003 #include <vector>
00004 #include <map>
00005 #include "Protocols.h"
00006 #include "Sensors.h"
00007 #include <Windows.h>
00008 #include <thread>
00009 #include <fstream>
00010 #include <chrono>
00011
00012 using namespace std;
00013
00019 class DriverAssistance
00020 {
00021 private:
00022     double* timestamp;
00023 public:
00027     DriverAssistance();
00032     ofstream positionlog;
00036     ofstream sensorprefusionlog;
00040     void launch(vector<Sensors*> _AudiVehicleSensors,vector<double*> _position, double* _drivetime, double
      _duration);
00044     void position_log_thread(vector<double*> _position, double* _drivetime, double _duration);
00048     void clock_thread_function();
00052     int SensorPreFusion(vector<Sensors*> _AudiVehicleSensors, double* _drivetime, double _duration);
00053     virtual ~DriverAssistance();
00054 };
00055
```

5.11 TecTask_Oenay_Can/ECU.cpp File Reference

```
#include "ECU.h"
```

5.12 ECU.cpp

```
00001 #include "ECU.h"
00002
00003
00004
00005 ECU::ECU()
00006 {
00007     //sense2recieve2this.insert(make_pair(0, 1));
00008 }
00009
00010 void ECU::run(vector<double*> _position,double* _drivetime, double _duration)
00011 {
00012     RunTimeEnvironment.launch(AudiVehicleSensors, _position,
      _drivetime, _duration);
00013 }
00014
00015 void ECU::SetPhysicalConnections(int _protocolid, int _sensorid) {
00016     sense2recieve2this.insert(make_pair(_protocolid, _sensorid));
00017 }
00018
00019 ECU::~ECU()
00020 {
00021 }
```

5.13 TecTask_Oenay_Can/ECU.h File Reference

```
#include <vector>
#include <map>
```

```
#include "Protocols.h"
#include "Sensors.h"
#include "DriverAssistance.h"
#include <iostream>
#include <string>
#include <chrono>
#include <windows.h>
#include <thread>
```

Classes

- class ECU

Typedefs

- typedef unsigned int uint32

5.13.1 Detailed Description

Contains only the ECU class

Definition in file ECU.h.

5.13.2 Typedef Documentation

5.13.2.1 uint32

```
typedef unsigned int uint32
```

Definition at line 14 of file ECU.h.

5.14 ECU.h

```
00001 #pragma once
00002 #include<vector>
00003 #include<map>
00004 #include "Protocols.h"
00005 #include "Sensors.h"
00006 #include "DriverAssistance.h"
00007 #include <iostream>
00008 #include <string>
00009 #include <chrono>
00010 #include <windows.h>
00011 #include <thread>
00012
00013 using namespace std;
00014 typedef unsigned int uint32;
00015
00022 class ECU
00023 {
00024 private:
00025     map <int, int> sense2recieve2this;
00026     DriverAssistance RunTimeEnvironment;
00027 public:
00031     ECU();
00035     vector<Sensors*> AudiVehicleSensors;
00039     vector<Protocols*> AudiVehicleProtocols;
00043     void SetPhysicalConnections(int _protocolid, int _sensorid);
00047     void run(vector<double*> _position,double* _drivetime, double _duration);
00048     virtual ~ECU();
00049 };
00050
```

5.15 TecTask_Oenay_Can/Header.h File Reference

```
#include "AudiVehicle.h"
```

5.16 Header.h

```
00001 #pragma once
00002 //#include "Sensors.h"
00003 //#include "Protocols.h"
00004 //#include "ECU.h"
00005 #include "AudiVehicle.h"
```

5.17 TecTask_Oenay_Can/miscellaneous.cpp File Reference

```
#include "miscellaneous.h"
```

Functions

- string tabs (int i)
- int Ltwnorm (vector< type_scaled_obj_coords > c)

5.17.1 Function Documentation

5.17.1.1 Ltwnorm()

```
int Ltwnorm (
            vector< type_scaled_obj_coords > c )
```

Definition at line 9 of file miscellaneous.cpp.

5.17.1.2 tabs()

```
string tabs (
            int i )
```

Definition at line 4 of file miscellaneous.cpp.

5.18 miscellaneous.cpp

```
00001 #include "miscellaneous.h"
00002
00003
00004 string tabs(int i) {
00005     string staff(i, '\t');
00006     return staff;
00007 }
00008
00009 int Ltwnorm(vector<type_scaled_obj_coords> c) {
00010     int ret = 0;
00011     for (unsigned int i = 0; i < c.size(); i++)
00012         ret += int(pow(c[i], 2));
00013     return ret;
00014 }
```

5.19 TecTask_Oenay_Can/miscellaneous.h File Reference

```
#include <iostream>
#include <string>
#include <vector>
```

Typedefs

- typedef unsigned short int type_scaled_obj_coords

Functions

- string tabs (int i)
- int Ltwnorm (vector< type_scaled_obj_coords > c)

5.19.1 Detailed Description

Containsglobal functions which may save a little more spacess.

Definition in file miscellaneous.h.

5.19.2 Typedef Documentation

5.19.2.1 type_scaled_obj_coords

```
typedef unsigned short int type_scaled_obj_coords
```

Definition at line 9 of file miscellaneous.h.

5.19.3 Function Documentation

5.19.3.1 Ltwnorm()

```
int Ltwnorm (
            vector< type_scaled_obj_coords > c )
```

Definition at line 9 of file miscellaneous.cpp.

5.19.3.2 tabs()

```
string tabs (
            int i )
```

Definition at line 4 of file miscellaneous.cpp.

5.20 miscellaneous.h

```
00001 #pragma once
00002 #include <iostream>
00003 #include <string>
00004 #include <vector>
00005
00008 using namespace std;
00009 typedef unsigned short int type_scaled_obj_coords;
00010
00011 string tabs(int i);
00012 int Ltwnorm(vector<type_scaled_obj_coords> c);
00013
```

5.21 TecTask_Oenay_Can/ObjEnvironment.cpp File Reference

```
#include "ObjEnvironment.h"
```

5.22 ObjEnvironment.cpp

```
00001 #include "ObjEnvironment.h"
00002
00003
00004
00005 Environment::Environment(string filename)
00006 {
00007     string line;
00008     ifstream myfile(filename);
00009     if (myfile.is_open())
00010     {
00011         while (getline(myfile, line))
00012         {
00013             stringstream stream(line);
00014             vector<double> current_real_object_coordinates(3);
00015             for (int i = 0; i < 3; i++) {
00016                 stream>>current_real_object_coordinates[i];
00017                 //cout << line[i]<<endl;
00018                 //cout << current_real_object_coordinates[i] << endl;
00019             }
00020             real_object_coordinates.push_back(current_real_object_coordinates);
00021             int current_type;
00022             stream >> current_type;
00023             real_object_type.push_back(current_type);
00024         }
00025         myfile.close();
00026     }
00027 }
00028
00029
00030 Environment::~Environment()
00031 {
00032 }
```

5.23 TecTask_Oenay_Can/ObjEnvironment.h File Reference

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
```

Classes

- class Environment

5.23.1 Detailed Description

Contains only the Environment Class. Attention, this class is not the sofware rte, yet the real environment.

Definition in file ObjEnvironment.h.

5.24 ObjEnvironment.h

```
00001 #pragma once
00002 #include <iostream>
00003 #include <vector>
00004 #include <string>
00005 #include <fstream>
00006 #include <sstream>
00007
00008 using namespace std;
00009
00017 class Environment
00018 {
00019 public:
00020     vector<vector<double>> real_object_coordinates;
00021     vector<int> real_object_type;
00022     Environment(string filename);
00023     ~Environment();
00024 };
00025
```

5.25 TecTask_Oenay_Can/Protocols.cpp File Reference

```
#include "Protocols.h"
#include <iostream>
```

5.26 Protocols.cpp

```
00001 #include "Protocols.h"
00002 #include<iostream>
00003
00004 using namespace std;
00005
00006
00007
00008 Protocols::Protocols()
00009 {
00010 }
00011
00012 Protocols::Protocols(int _id, bool _on_off, string _type)
00013 {
00014     id = _id;
00015     on_off = _on_off;
00016     type = _type;
00017 }
00018
00019 bool Protocols::get_status()
00020 {
00021     return on_off;
00022 }
00023
00024 string Protocols::get_type()
00025 {
00026     return type;
00027 }
00028
00029 int Protocols::get_id() {
00030     return id;
00031 }
00032
00033 void Protocols::print_this()
00034 {
00035     cout << tabs(1) << " has a type of " << type << " protocoll"
00036         << " and has a status of " << on_off << endl;
00037 }
00038
00039 Protocols::~Protocols()
00040 {
00041 }
00042
00043 RadarEthernet::RadarEthernet(int _id, bool _onoff, int _bits_per_ms) :
      Protocols(_id, _onoff,"RadarEthernet"), bits_per_ms(_bits_per_ms){}
```

```
00044
00045 void RadarEthernet::print_this() {
00046     cout <<tabs(1)<<" has a type of " <<type<<" protocoll"
00047     << " which has a rate of communication " << bits_per_ms
00048     << " and has a status of " <<on_off<< endl;
00049 };
```

5.27 TecTask_Oenay_Can/Protocols.h File Reference

```
#include <vector>
#include "miscellaneous.h"
```

Classes

- class Protocols
- class RadarEthernet

5.27.1 Detailed Description

Contains only the parent protocol class and the child radar ethernet class

Definition in file Protocols.h.

5.28 Protocols.h

```
00001 #pragma once
00002 #include <vector>
00003 #include "miscellaneous.h"
00004 using namespace std;
00010 class Protocols
00011 {
00012 protected:
00013     int id;
00014     bool on_off;
00015     string type;
00016 public:
00017     Protocols();
00018     Protocols(int _id, bool _on_off = false, string _type="None");
00019     virtual bool get_status();
00020     virtual string get_type();
00021     virtual int get_id();
00022     virtual int get_bits_per_ms() { return 0; };
00023     virtual void print_this();
00024     virtual ~Protocols();
00025 };
00026
00027 class RadarEthernet : public Protocols {
00028 private:
00029     int bits_per_ms;
00030 public:
00031     RadarEthernet(int _id, bool _onoff = false, int _bits_per_ms = 0);
00032     virtual int get_bits_per_ms() { return bits_per_ms; };
00033     virtual int get_id() { return id; };
00034     virtual void print_this();
00035 };
```

5.29 TecTask_Oenay_Can/Sensors.cpp File Reference

```
#include "Sensors.h"
#include <iostream>
#include <fstream>
#include <string>
#include <thread>
#include <Windows.h>
#include "ObjEnvironment.h"
```

5.30 Sensors.cpp

```
00001 #include "Sensors.h"
00002 #include <iostream>
00003 #include <fstream>
00004 #include <string>
00005 #include <thread>
00006 #include <Windows.h>
00007 #include "ObjEnvironment.h"
00008
00009 using namespace std;
00010
00011
00012
00013 Sensors::Sensors()
00014 {
00015 }
00016
00017 Sensors::Sensors(int _id, vector<double> _pos, string _envfile, bool initial_off, string
      _type)
00018 {
00019     env=new Environment(_envfile);
00020
00021
00022     id = _id;
00023     for (int i = 0; i < 3; i++)
00024         pos.push_back(_pos[i]);
00025     on_off = initial_off;
00026     type = _type;
00027 }
00028
00029 bool Sensors::get_status()
00030 {
00031     return on_off;
00032 }
00033
00034 string Sensors::get_type()
00035 {
00036     return type;
00037 }
00038
00039 int Sensors::get_id()
00040 {
00041     return id;
00042 }
00043
00044
00045 void Sensors::print_this()
00046 {
00047     cout << tabs(1) << " has a type of " << type << " sensor"
00048     << " and has a status of " << on_off << endl;
00049 }
00050
00051 Sensors::~Sensors()
00052 {
00053 }
00054
00055 RadarSensor::RadarSensor(int _id, vector<double> _pos, vector<double> _range,
      string _envfile,
00056                          bool _onoff, type_timestamp _timestamp,
      type_objectclass _objectClass)
00057     : Sensors(_id, _pos, _envfile, _onoff, "RadarSensor"), timestamp(_timestamp) , objectClass(
      _objectClass)
00058 {
00059     for(unsigned int i=0;i<_range.size();i++)
```

```
00060     range.push_back(_range[i]);
00061     r_pos.resize(3);
00062     obj_coords.resize(3);
00063     for (int i = 0; i < 3; i++)
00064     {
00065         r_pos[i] = 0;
00066         obj_coords[i] = 0;
00067     }
00068 }
00069
00070 void RadarSensor::print_this() {
00071     cout << tabs(1) << " has a type of " << Sensors::type<<" sensor"
00072     << " which has a timestamp of " << timestamp
00073     << " position of [X:"<<pos[0]<<" Y:"<<pos[1]<<" Z:"<<pos[2]<<"]"
00074     << " has range of "<<range[0]<<" l-r "<<range[1]<<" f-a"
00075     << " and has a status of " << Sensors::on_off<< endl;
00076     };
00077
00078 void RadarSensor::SenseObjEnvironment(vector<double*> _position, double*
    _drivetime, double _duration) {
00079     catchlog.open("Threadlog_catches_of_RadarSensor_"+to_string(id)+".log", ofstream::out);
00080     while (*_drivetime<_duration) {
00081         Sleep(1);
00082         for (int i = 0; i < 3; i++)
00083             r_pos[i] = pos[i] + *_position[i];
00084         timestamp = type_timestamp(*_drivetime);
00085         for (unsigned int j = 0; j < env->real_object_coordinates[0].size(); j++)
00086         {
00087             double difx = env->real_object_coordinates[0][0] -
    r_pos[0];
00088             double dify = env->real_object_coordinates[0][1] -
    r_pos[1];
00089             if (difx < range[0] && difx > -range[0] && dify < range[1] && dify > -
    range[1])
00090             {
00091
00092                 objectClass = type_objectclass(env->
    real_object_type[0]);
00093                 for (int i = 0; i < 3; i++)
00094                 {
00095                     obj_coords[i] = type_unscaled_obj_coords(
    env->real_object_coordinates[0][i] - *_position[i]);
00096                     obj_coords[i] = obj_coords[i] << 1000;
00097                 }
00098                 //some charachters are unvisible therefore further standard unsigned shall be used here
00099                 //catchlog << timestamp << " " << unsigned(objectClass) << endl;
00100                 catchlog << *_drivetime << " " << env->
    real_object_type[0] << endl;
00101             }
00102             else
00103             {
00104                 objectClass = type_objectclass(0);
00105                 for (int i = 0; i < 3; i++)
00106                     obj_coords[i] = type_unscaled_obj_coords(0);
00107
00108                 catchlog << *_drivetime << " " << 0 << endl;
00109             }
00110         }
00111
00112     }
00113     catchlog.close();
00114
00115 }
```

5.31 TecTask_Oenay_Can/Sensors.h File Reference

```
#include <vector>
#include <thread>
#include "miscellaneous.h"
#include "ObjEnvironment.h"
```

Classes

- class Sensors
- class RadarSensor

Typedefs

- typedef unsigned short int type_unscaled_obj_coords
- typedef unsigned short int type_scaled_obj_coords
- typedef unsigned int type_timestamp
- typedef uint8_t type_objectclass

5.31.1 Detailed Description

Contains only the parent sensor class and the child radar sensor class

Definition in file Sensors.h.

5.31.2 Typedef Documentation

5.31.2.1 type_objectclass

```
typedef uint8_t type_objectclass
```

Definition at line 10 of file Sensors.h.

5.31.2.2 type_scaled_obj_coords

```
typedef unsigned short int type_scaled_obj_coords
```

Definition at line 8 of file Sensors.h.

5.31.2.3 type_timestamp

```
typedef unsigned int type_timestamp
```

Definition at line 9 of file Sensors.h.

5.31.2.4 type_unscaled_obj_coords

```
typedef unsigned short int type_unscaled_obj_coords
```

Definition at line 7 of file Sensors.h.

5.32 Sensors.h

```
00001 #pragma once
00002 #include <vector>
00003 #include <thread>
00004 #include "miscellaneous.h"
00005 #include "ObjEnvironment.h"
00006
00007 typedef unsigned short int type_unscaled_obj_coords;
00008 typedef unsigned short int type_scaled_obj_coords;
00009 typedef unsigned int type_timestamp;
00010 typedef uint8_t type_objectclass;
00011
00012 //cout << sizeof(unsigned short int) * 8 << endl; // for relative coordinates
00013 //cout << sizeof(unsigned int)*8 << endl; // for timestamp
00014 //cout << sizeof(unsigned __int8) * 8 << endl; //for objectclass
00015
00022 using namespace std;
00023 class Sensors
00024 {
00025 protected:
00026     int id;
00027     bool on_off;
00028     string type;
00029     vector<double> pos;
00030     Environment* env;
00031 public:
00032     Sensors();
00033     Sensors(int _id, vector<double> _pos, string _envfile, bool initial_off = false, string _type =
    "None" );
00034     virtual bool get_status();
00035     virtual string get_type();
00036     virtual int get_id();
00037     virtual void print_this();
00038     virtual void SenseObjEnvironment(vector<double*> _position, double* _drivetime,
    double _duration) {};
00039     virtual type_timestamp Get_timestamp() { return 0; };
00040     virtual type_objectclass Get_objectClass() { return 0; };
00041     virtual vector<type_unscaled_obj_coords> Get_objectCoords() {
    vector<type_unscaled_obj_coords> dummy; return dummy; };
00042     bool operator <(const Sensors& rhs) const {return id < rhs.id;}
00043     virtual ~Sensors();
00044 };
00045
00046 class RadarSensor : public Sensors {
00047 private:
00048     vector <type_unscaled_obj_coords> obj_coords;
00049     type_timestamp timestamp;
00050     type_objectclass objectClass;
00051     vector <double> range;
00052     vector<double> r_pos;
00053 public:
00054     RadarSensor(int _id,vector<double> _pos, vector<double> _range, string _envfile, bool _onoff
     = false, type_timestamp _timestamp=0, type_objectclass _objectClass=0);
00055     ofstream catchlog;
00056     virtual int get_id() { return id; }
00061     virtual void SenseObjEnvironment(vector<double*> _position, double* _drivetime, double _duration);
00065     virtual type_timestamp Get_timestamp() { return timestamp; };
00069     virtual type_objectclass Get_objectClass() { return objectClass; };
00073     virtual vector<type_unscaled_obj_coords> Get_objectCoords() { return obj_coords; };
00074     virtual void print_this();
00075 };
00076
```

5.33 TecTask_Oenay_Can/stdafx.cpp File Reference

```
#include "stdafx.h"
```

5.34 stdafx.cpp

```
00001 // stdafx.cpp : Quelldatei, die nur die Standard-Includes einbindet.
00002 // test1.pch ist der vorkompilierte Header.
```

```
00003 // stdafx.obj enthält die vorkompilierten Typinformationen.
00004
00005 #include "stdafx.h"
00006
00007 // TODO: Auf zusätzliche Header verweisen, die in STDAFX.H
00008 // und nicht in dieser Datei erforderlich sind.
```

5.35 TecTask_Oenay_Can/stdafx.h File Reference

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
```

5.36 stdafx.h

```
00001 // stdafx.h: Includedatei für Standardsystem-Includedateien
00002 // oder häufig verwendete projektspezifische Includedateien,
00003 // die nur in unregelmäßigen Abständen geändert werden.
00004 //
00005
00006 #pragma once
00007
00008 #include "targetver.h"
00009
00010 #include <stdio.h>
00011 #include <tchar.h>
00012
00013
00014
00015 // TODO: Hier auf zusätzliche Header, die das Programm erfordert, verweisen.
```

5.37 TecTask_Oenay_Can/targetver.h File Reference

```
#include <SDKDDKVer.h>
```

5.38 targetver.h

```
00001 #pragma once
00002
00003 // Durch Einbeziehen von"SDKDDKVer.h" wird die höchste verfügbare Windows-Plattform definiert.
00004
00005 // Wenn Sie die Anwendung für eine frühere Windows-Plattform erstellen möchten, schließen Sie "WinSDKVer.h"
       ein, und
00006 // legen Sie das _WIN32_WINNT-Makro auf die zu unterstützende Plattform fest, bevor Sie "SDKDDKVer.h"
       einschließen.
00007
00008 #include <SDKDDKVer.h>
```