European Commission

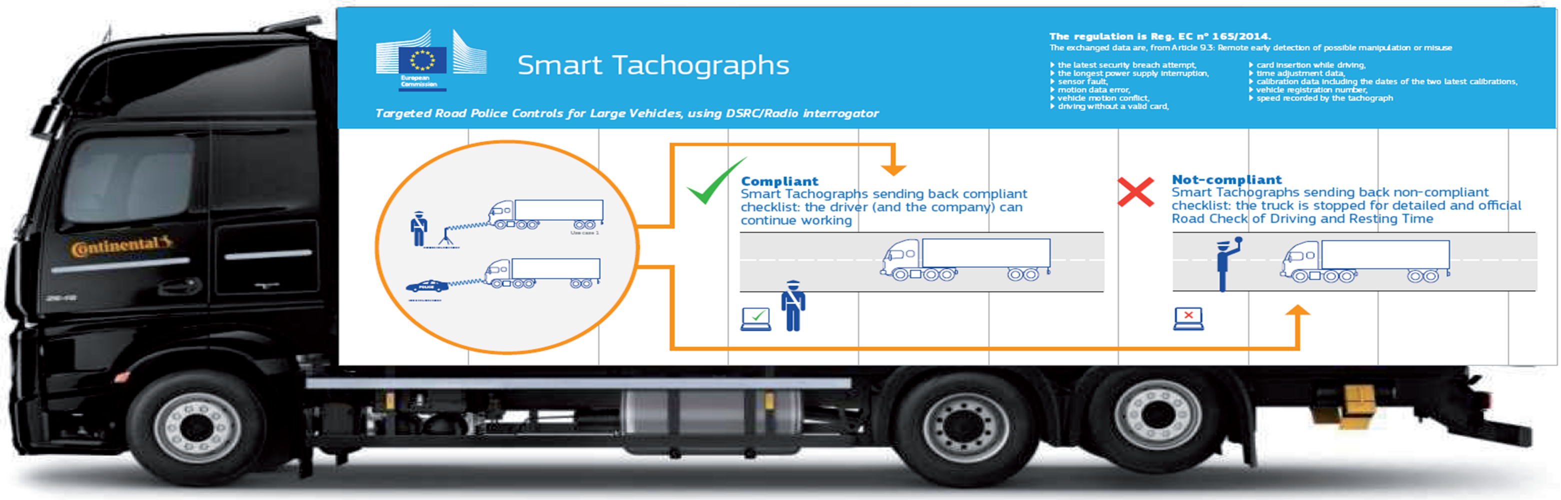
JRC TECHNICAL REPORTS

# Proof of Concept (PoC) of the remote interrogation for the smart tachograph based on CEN-Dedicated Short Range Communications (DSRC)

*Description of the CEN-DSRC prototype for remote interrogation*

Gianmarco Baldini,
Raimondo Giuliani,
Eduardo Cano-Pons

2016


Smart Tachographs
Targeted Road Police Controls for Large Vehicles, using DSRC/Radio interrogator
The regulation is Reg. EC n° 165/2014.
The exchanged data are, from Article 9.3: Remote early detection of possible manipulation or misuse
the latest security breach attempt,
the longest power supply interruption,
sensor fault,
motion data error,
vehicle motion conflict,
driving without a valid card,
card insertion while driving,
time adjustment data,
calibration data including the dates of the two latest calibrations,
vehicle registration number,
speed recorded by the tachograph
Compliant
Smart Tachographs sending back compliant checklist: the driver (and the company) can continue working
Not-compliant
Smart Tachographs sending back non-compliant checklist: the truck is stopped for detailed and official Road Check of Driving and Resting Time
Continental

Joint Research Centre

EUR 28378 EN

This publication is a Technical report by the Joint Research Centre (JRC), the European Commission's science and knowledge service. It aims to provide evidence-based scientific support to the European policymaking process. The scientific output expressed does not imply a policy position of the European Commission. Neither the European Commission nor any person acting on behalf of the Commission is responsible for the use that might be made of this publication.

**Contact Information**
Name: Gianmarco Baldini
Address: European Commission, Joint Research Centre, Via Enrico Fermi 2749, TP 360, 21027 Ispra (VA), Italy
E-mail: gianmarco.baldini@jrc.ec.europa.eu
Tel.: +39 0332 78 6618

**JRC Science Hub**
https://ec.europa.eu/jrc

JRC105039

EUR 28378 EN

| | | | |
|---|---|---|---|
| PDF | ISBN 978-92-79-64805-2 | ISSN 1831-9424 | doi:10.2788/866989 |
| Print | ISBN 978-92-79-64804-5 | ISSN 1018-5593 | doi:10.2788/422399 |

Luxembourg: Publications Office of the European Union, 2016

How to cite: Gianmarco Baldini et al. Proof of concept for remote early interrogation. EUR 28378 EN. doi 10.2788/866989

All images © European Union 2016

**PoC of the remote interrogation for the smart tachograph based on CEN-DSRC**
This report provides an overview of Proof of Concept for remote interrogation for the smart tachograph application

# PoC of the remote interrogation for the smart tachograph based on CEN-DSRC

Gianmarco Baldini,

Raimondo Giuliani,

Eduardo Cano-Pons

# Contents

# Abstract

The aim of this technical report is to describe the proof-of-concept of the CEN-DSRC implementation of the remote interrogation function of the new version of the digital tachograph. The current digital tachograph (DT) system to monitor the driving time in commercial vehicles above 3.5 tons is governed by Council Regulation (EEC) No 3821/85 of 20 December 1985, which was modified at several occasions and more recently in 2006, when the digital tachograph was introduced, and in 2009, when it was updated to technical progress to avoid fraud and reduce the administrative burden. In July 2011 the Commission made a proposal (COM(2011) 451 final) to modify the tachograph regulation, which has been the object of discussions in Council and Parliament in the course of the ordinary legislative procedure. The final version of the approved regulation was published in February 2014 (Regulation 165/2014 (Commission, 2004)). The technical specifications of the smart tachograph were published as Regulation 799/2016 (Commission, 2011). One of the main functions is the remote interrogation of the Smart Tachograph (ST) installed in the commercial vehicle through the CEN-DSRC standard. The function supports law enforcers in the checking of potential frauds or malfunctions in the ST. To support the future deployment of the smart tachograph and to validate the technical specifications of the smart tachograph regarding the remote interrogation function, JRC issued a tender to the DSRC manufacturer Q-FREE to implement a prototype of the new remote interrogation systems. Q-FREE was chosen because it is only the leading producer of CEN-DSRC equipment and because it was not directly involved in the discussion of the technical specifications of the ST, so it did not have specific bias and it could provide a critical review of the specifications. The prototype was successfully implemented and tested. It was shown during the JRC Open day in May 2016 to thousands of visitors at the stand of the smart tachograph organized by unit DG.JRC.E3.

# 1. Introduction

The current digital tachograph (DT) system to monitor the driving time in commercial vehicles above 3.5 tons is governed by Council Regulation (EEC) No 3821/85 of 20 December 1985, which was modified at several occasions and more recently in 2006, when the digital tachograph was introduced, and in 2009, when it was updated to technical progress to avoid fraud and reduce the administrative burden. In July 2011 the Commission made a proposal (COM(2011) 451 final) to modify the tachograph regulation, which has been the object of discussions in Council and Parliament in the course of the ordinary legislative procedure. The final version of the approved regulation was published in February 2014 (Regulation 165/2014) (Commission, 2004). From the publication date of the new regulation, the technical specifications of the new digital tachograph must be defined within a time frame of 24 months (February 2016). According to the new regulation, the Tachograph shall be equipped with a remote communication functionality that shall allow law enforcers to read Tachograph information from passing vehicles at a road side control site.

One of the limitations of the current version of the Digital Tachograph is that the law enforcer must stop each commercial vehicle to perform an inspection. Obviously the number of commercial vehicles, which can be inspected in such a fashion is limited. In the drafting of the regulation (EU) NÂř1652014 (Commission, 2004), it was proposed to provide means for targeted interrogation of the digital tachograph in a commercial vehicle to filter out potential infringements of the regulation. From an operational point of view, this means that a law enforcer will be able to interrogate directly the smart tachograph in a moving truck using an enforcement wireless communication equipment. Note that is not the intention of the legislator that the data transmitted thru the wireless communication will be used for direct fining, but it is only a selection tool to stop commercial vehicles and subsequently perform a complete manual check.

During the technical discussion with the stakeholders involved in the revision of the Digital Tachograph, CEN-Dedicated Short Range Communications (DSRC) has been selected as the main wireless communication technology to be used for the remote communication functionality. The exchanged data shall contain only those required for targeted road side checks, which is defined in the Article 9 of the published regulation 165/2014.

Even if the CEN-DSRC standard is widely deployed and various studies have already been performed for the application of electronic tolling, the use cases defined for the new DT can be quite different. The two use cases are a) roadside check with a CEN-DSRC reader operated by law enforcers on the side of the road and b) mobile reader installed in a law enforcer vehicle. In addition, the size of the data to be exchanged is also different from what is defined in electronic tolling.

To support the future deployment of the smart tachograph and to validate the technical specifications of the smart tachograph regarding the remote interrogation function, JRC issued a tender to the DSRC manufacturer Q-FREE to implement a prototype of the new remote interrogation systems. Q-FREE was chosen because it is only the leading producer of CEN-DSRC equipment and because it was not directly involved in the discussion of the technical specifications of the Smart Tachograph (ST), so it did not have specific bias and it could provide a critical review of the specifications. The prototype was successfully implemented and tested. It was shown during the JRC Open day in May 2016 to thousands of visitors at the stand of the smart tachograph organized by unit DG.JRC.E3.

This report provide a simple description of the remote interrogation function in the fugure smart tachograph and a description of the proof of concept and its presentation at the JRC Open Day in May 2016 in front of thousands of visitors.

## 2. Remote Interrogation

The goal of the remote communication as described in (Commission, 2004) and (Commission, 2011) determines that the tachograph shall be equipped with a remote communication functionality that shall enable agents of the competent control authorities to read tachograph information from passing commercial vehicles by using remote communication equipment. This equipment is called the Remote early detection communication reader. It is important to comprehend that this functionality is intended to serve only as a pre-filter in order to select vehicles for closer inspection, and it does not replace the formal inspection process as determined in the provisions of Regulation (EU) No. 165/2014 (recital 9 in the preamble of this regulation (Commission, 2004), stating that remote communication between the tachograph and control authorities for roadside control purposes facilitates targeted roadside checks).

The regulation requests a very precise set of data, which is described here:

1. the *latest security breach attempt*. This is the latest security breach attempt recorded by the system, which gives a clear indication that a malicious entity has tampered with the system.
2. the *longest power supply interruption*. This is the longest interruption of the power supply, which may give indication on the potential manipulation of the tachograph by a malicious entity.
3. *sensor fault*, which gives indication on a fault of the motion sensor or the Global Navigation Satellite System (GNSS) sensor.
4. *motion data error*, which provides an indication of potential errors in the processing of the data from the motion sensor.
5. *vehicle motion conflict*, which indicates a discrepancy between the position or speed recorded by the motion sensor, the GNSS sensor or any other sensor used by the manufacturers. This event is an important information, which could be used to detect malicious tampering of the tachograph.
6. *driving without a valid card*, which gives a direct information that there is non compliance to the regulation.
7. *card insertion while driving*,
8. *time adjustment data*. This information identifies the moment when the tachograph needed to readjust the time. This information is also useful to detect a malicious activity because tampering with the time of the tachograph could give an economic benefit to a criminal driver or company.
9. *calibration data including the dates of the two latest calibrations*. The calibration data is needed to ensure that calibration time is consistent with the operation of the vehicle and the tachograph.
10. *vehicle registration number*, which identifies the vehicle and the Vehicle Unit (VU).
11. *speed recorded by the tachograph*. This information cannot be directly used to detect infringements against speed limits, but it can be used to detect malfunctions (intentional or un-intentional) in the speed calculation and recording of the tachograph. We note that the speed can be calculated both through the GNSS receiver and the odometer.

These data has been defined in the regulatory process for filtering purpose and to highlight the possibility of tampering or malfunction (e.g., the events field). The evaluation for the conformation to the regulation is quite a complex process, which require access to most of the recorded data of the tachograph in the last 28 days of operation. This evaluation could not be implemented and executed on the road, but it requires the stop of the vehicle, the download of data and the analysis of the data with special software. On the other side, the current version of the digital tachograph has been subject to various attacks to undermine the integrity of the collected (i.e., from the motion sensor) data or the recorded data as described in the introduction section **??**.

In the current version of the digital tachograph, a law enforcer must stop the vehicle to check the presence of events, which could indicate malfunction or tampering. In the new version of the Smart Tachograph, the remote communication through the CEN-DSRC at 5.8 GHz can provide the list of outstanding events and other useful information, which can alert the law enforcer for potential misuse of the smart tachograph application.

The typical scenarios where the CEN-DSRC can be used are shown in figure 1, where the smart tachograph present in the commercial vehicle can be interrogated through CEN-DSRC either from a mobile vehicle or a roadside equipment system. This proof of concept has been mainly designed for the roadside use case, even it can be easily adapted to the mobile vehicle as well.

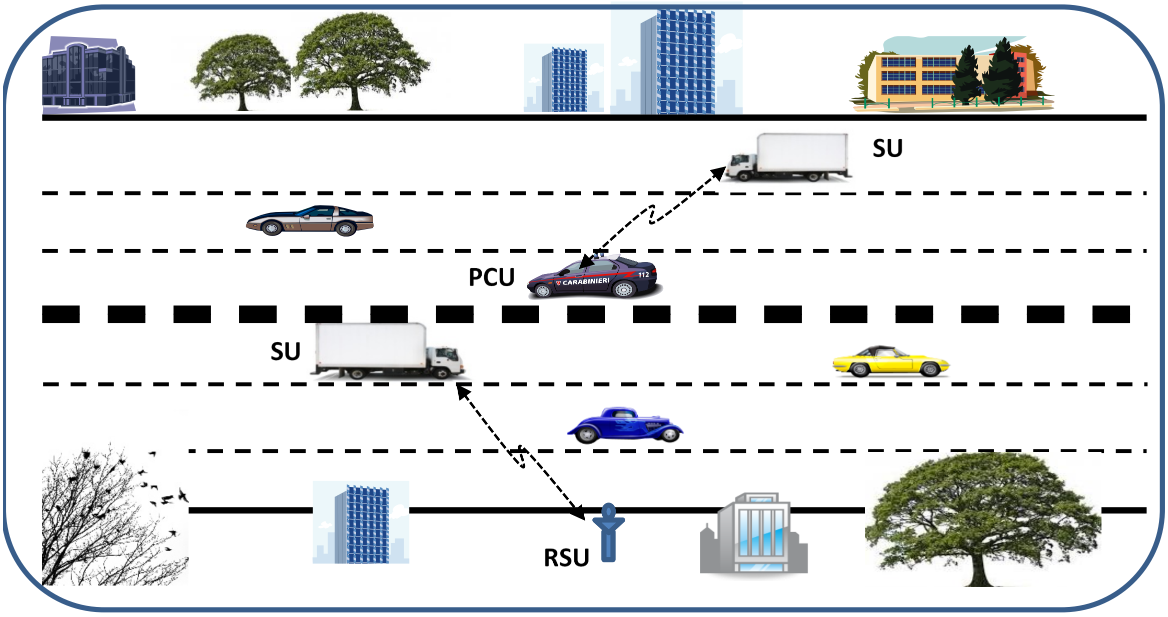


**Figure 1:** Typical scenarios for the application of CEN-DSRC to the smart tachograph

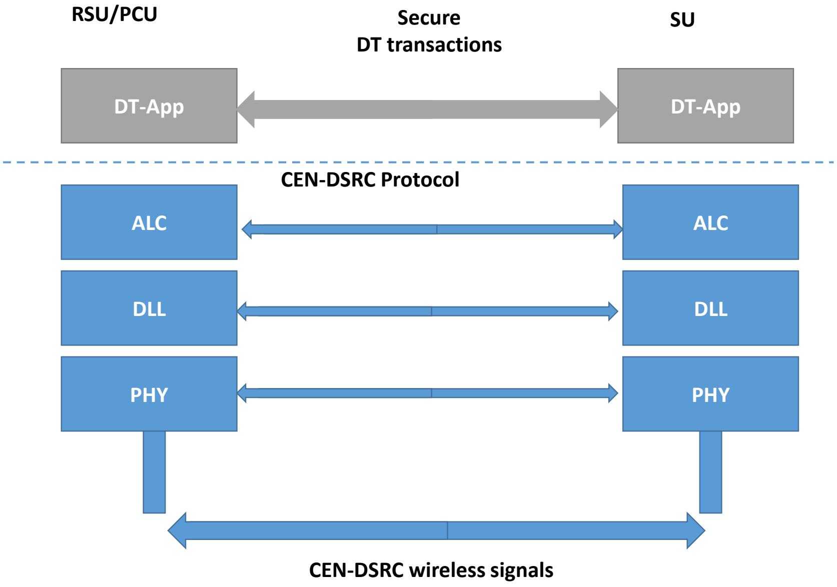


**Figure 2:** remote communication architecture

In the rest of this section, we describe in detail how the remote communication function is implemented in the smart tachograph. The overall architecture of the smart tachograph for the specific aspects of remote communication are described in figure 3.

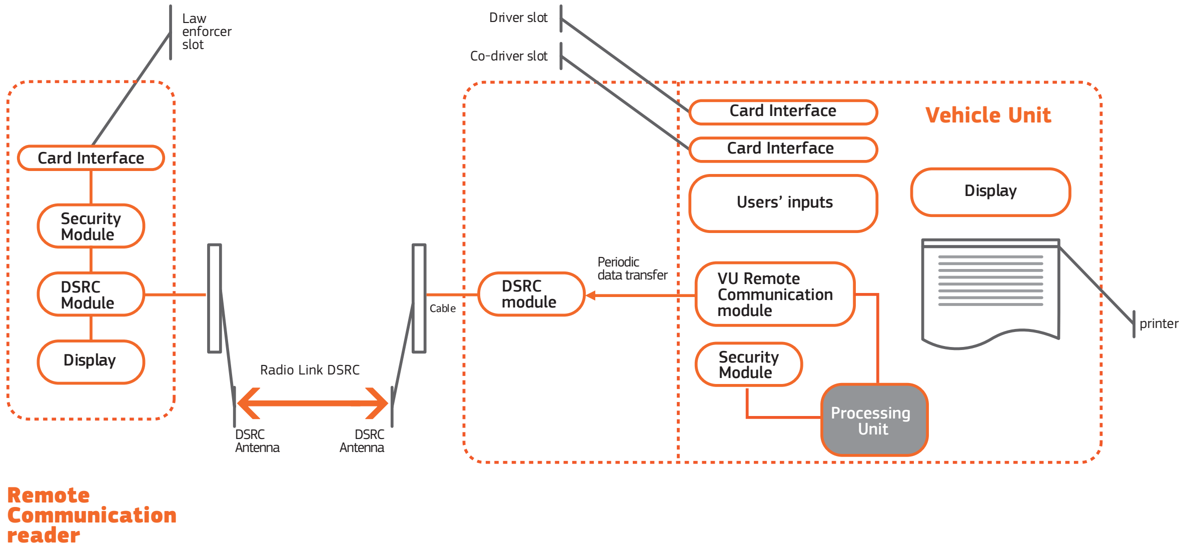


**Figure 3:** remote communication architecture

The system on the left of figure 3 represent the reader system used by the law enforcer on the road or from a mobile police vehicle. The law enforcer activates the system by inserting his/her smartcard. This operation is needed to authenticate the law enforcer. After authentication, a law enforcer can interrogate a commercial vehicle by requesting the remote communication reader to issue a wireless CEN-DSRC challenge to the DSRC On Board Unit (OBU) in the commercial vehicle by using the CEN-DSRC standard at 5.8 GHz. The challenge specifies the applications for which the law enforcer requests the data. Each application has its own id. At the moment, only one application is defined: the Smart Tachograph, but other applications can also be implemented in the DSRC OBU. In fact, Appendix 14 of the technical specifications of the Smart Tachograph specifies the support for another regulated application: the Weighing and Dimensions directive, which also uses the CEN-DSRC 5.8 GHz. For this application, the use of CEN-DSRC is explicitly requested. In the future the same system could support other applications. Each application is identified by the related application id. The wireless challenge contains the list of application id to which the DSRC OBU must provide data is it is available.

Upon the reception of the wireless challenge, the DSRC OBU checks the application id and verifies if the related data is present in the system. The process of storing the data in the DSRC OBU is described in the following paragraph. The VU of the Smart Tachograph periodically (every 60 seconds) verifies the content of its memory to verify the presence of new events, the update of status information, the current values of specific parameters and so on. The VU generates the data identified before in this section from its current memory. Before the information is stored in the DSRC-VU module, a message authentication code is appended to ensure its integrity and authenticity. Symmetric keys are used to secure the data to be sent in response to the wireless challenge request. This process does also implement authentication of the commercial vehicles, because it embeds the vehicle registration number of the commercial vehicle and the serial number of the VU. The authentication is needed to ensure that the VU has not been replaced by another VU, which could be used to provide false information.

In this way, end-to-end security is implemented, where the two ends are respectively the remote reader used by the law enforcer and the vehicle unit with its cryptomodule. It was a design decision to implement a new end-to-end security mechanism rather than leaning on the proprietary security solutions already defined in the electronic tolling standards ((**?**)). There were two reasons for this: the first one is for future upgradeability of the system for future wireless communication systems. In the long foreseen lifetime of the smart tachograph (15 years or more), new communication technologies could be developed. With end to end authentication, the CEN-DSRC at 5.8 GHz could be easily be replaced without an significant impact on the rest of the smart tachograph system. The second reason is that an harmonized security standard at European level must be defined, which would take time to develop, while the technical specifications needed to be finalized in a specific timeframe.

A specific workflow for the wireless CEN-DSRC has been defined (details are in (Commission, 2011)). This worflow is derived from existing applications like the electronic tolling, but it is has been improved and made more efficient for the specific needs of the smart tachograph (e.g., the type and format of data to be transmitted and the absence of fields, which are specific for electronic tolling). An unique workflow was implemented also because there is not an unique electronic tolling workflow across Europe. Countries like Italy has a different electronic tolling implementation in comparison to country like Germany or France. While the application layer workflow was specifically designed for the smart tachograph, the definition of the physical layer was strictly based on the standard EN 13372.

This was done for various reasons:

1. The need to use mass market electronic components, with a wide market deployment.
2. to be conformant to the radio frequency spectrum regulations already valid for the EN 12253.
3. to reuse the existing testing standards already defined for EN 12253.

In this way, the hardware implementations of the CEN-DSRC available in the market for electronic tolling, could also used for the smart tachograph with a new version of the software, implementing the challenge-response interaction between the reader and the OBU in the commercial vehicle. This provides the advantage of decreased costs for the smart tachograph.

A simplified schema of the overall workflow and layered stack of the CEN-DSRC communication is shown in figure fig:flowdsrc. The higher layer is the application layer where the information is exchanged. The application layer core (ALC) directly interacts with the DT Application for communications. The lower layers of the network are the the physical layer (PHY) and the data link layer (DLL) which implements the medium access control (MAC).

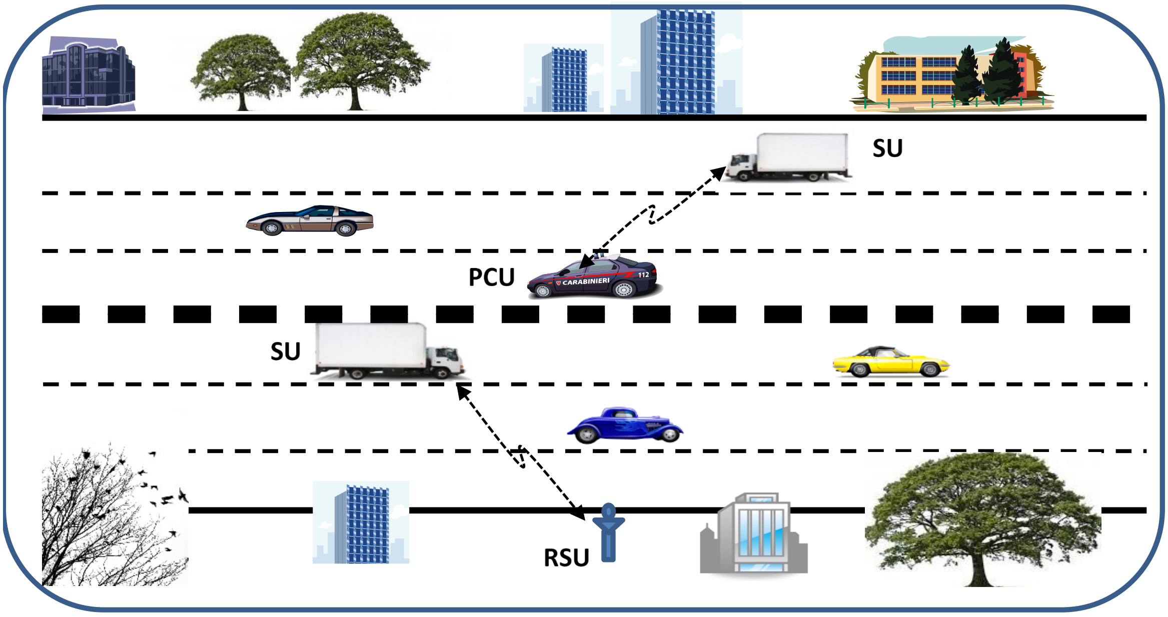


**Figure 4:** Layered protocol for CEN-DSRC in the new version of the Digital Tachograph

As described previously, the application layer and parts of the ALC have been implemented in this prototype. All the other sub-layers are implemented with the same hardware components of the european electronic tolling.

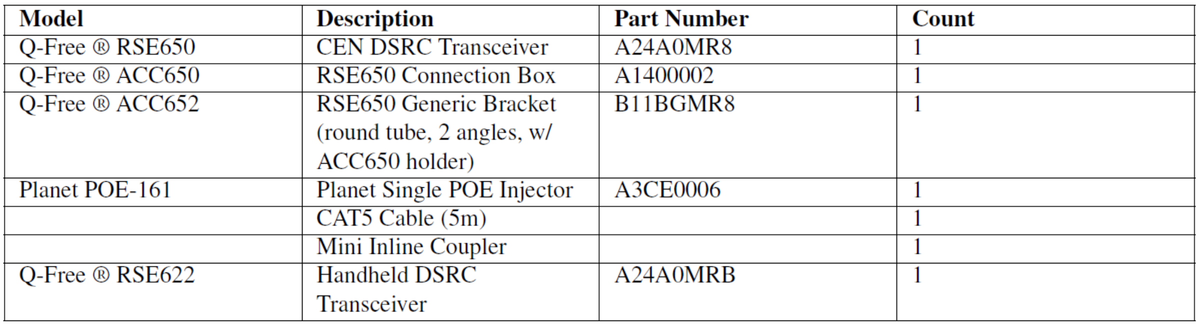
## 3. Proof-of-Concept

In 2016, a contract was given to Q-FREE to implement the new workflow of the remote interrogation of the smart tachograph on the basis of the technical specifications defined in (Commission, 2011).

The prototype was composed by a remote reader and four CEN-DSRC tags to be installed in the commercial vehicles.

Q-Free implemented the proof-of-concept with the following hardware and software components:

- the hardware systems already developed for electronic tolling: RSE650 for the remote reader equipment (one unit) and OBU615 for the OBU (four units). The RSE650 was provided together with auxiliary components and power supply as for the list of items provided in figure 5.
- the software was implemented on the basis of the specifications in (Commission, 2011). This was the main part of the work.



| Model | Description | Part Number | Count |
|---|---|---|---|
| Q-Free ® RSE650 | CEN DSRC Transceiver | A24A0MR8 | 1 |
| Q-Free ® ACC650 | RSE650 Connection Box | A1400002 | 1 |
| Q-Free ® ACC652 | RSE650 Generic Bracket (round tube, 2 angles, w/ ACC650 holder) | B11BGMR8 | 1 |
| Planet POE-161 | Planet Single POE Injector | A3CE0006 | 1 |
|  | CAT5 Cable (5m) |  | 1 |
|  | Mini Inline Coupler |  | 1 |
| Q-Free ® RSE622 | Handheld DSRC Transceiver | A24A0MRB | 1 |

**Figure 5:** List of hardware components for the reader system

The overall system was used during the Open Day in May 2016. Figure 6 shows the stand where the CEN-DSRC prototype reader was positioned during the Open Day. Behind the reader, the representatives of the Italian road enforcement (who are going to use the reader) can be seen. The participation of the Italian police force was quite useful to receive a feedback, which was quite positive on the new tool.

Figure 7 shows the images of the CEN-DSRC prototype while it was used to interrogate the CEN-DSRC OBU installed in a commercial vehicle provided by Continental for the JRC Open Day 2016. The prototype was able to interrogate and process the data many times a second. A spectrum analyzer was also used during the open day to evaluate the levels of transmitted Radio Frequency (RF) power by the CEN-DSRC system, which was compliant to the RF spectrum regulations.

The manual of use of the CEN-DSRC system is provided in the Annex 1. The manual of the software components of the CEN-DSRC system, their structure and how the software can be used is described in Annex 2.

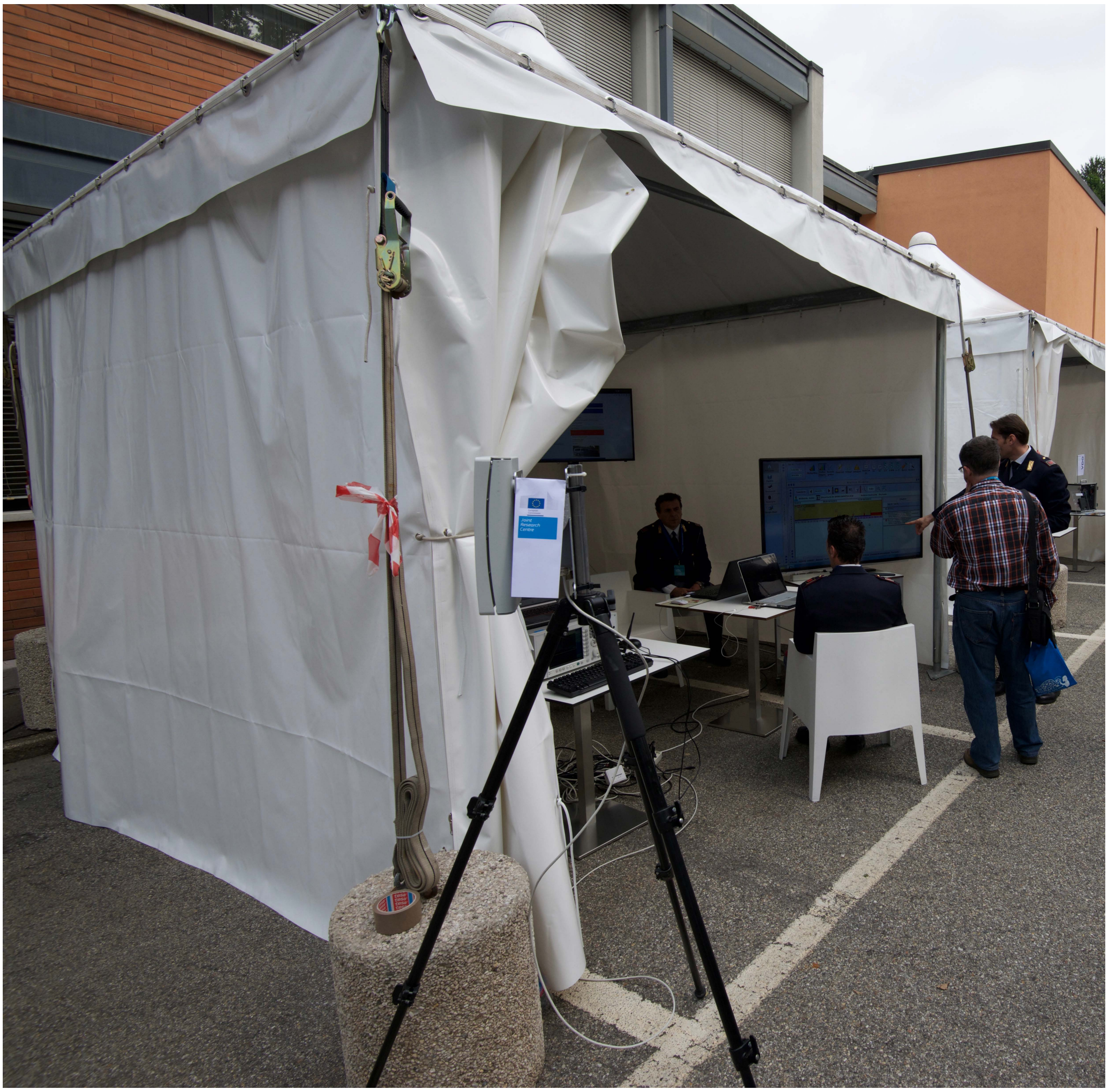


**Figure 6:** Image of the CEN-DSRC prototype reader during the JRC Open Day 2016

## 4. Conclusions

This report describes the prototype, which implements the remote interrogation function for the smart tachograph based on the CEN-DSRC standard. The prototype has been evaluated and tested by the JRC and it satisfies the operational requirements of the smart tachograph. It has been used and shown at the JRC Open Day in May 2016 in collaboration with the Italian law enforcement (i.e., Italian Polizia Stradale), which will be one of users of this tool together with their other European colleagues. The remote interrogation function will allow a more efficient filtering of the commercial vehicles on the road for the smart tachograph regulation.

**Figure 7:** Image of the CEN-DSRC prototype reader interrogating the CEN-DSRC OBU installed in a Continental truck

# 5. Annex 1 - Manual of use of the remote interrogator for the smart tachograph

# Tachograph Getting Started Guide

| | |
|---|---|
| DocumentID: | 2016-642-Tachograph-001 |
| Revision number: | 1 |
| Edited by: | Truls Gulbrandsen |
| Checked by: | Not checked |
| Approved by: | Not approved |

## Revision history:

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| v1.0 | 2016-03-01 | First revision of document | TMAG |

# Contents

## References

### USER DOCUMENTATION

- [1] 2009-645-RSE650-001 RSE650 installation manual
- [2] 2011-642-RSE650-mdmrapp-001 User Manual

## Glossary

**DSRC**
Dedicated Short-Range Communications

**OBU**
On-Board Unit

**RTM**
Remote Tachograph Monitoring

# 1 Introduction

This guide tells you how to quickly get started using the tachograph kit.

# 2 Kit

## 2.1 Hardware

Table 1 shows the hardware included in the kit.

Table 1: Hardware

| Model | Description | Part Number | Count |
|---|---|---|---|
| Q-Free ® RSE650 | CEN DSRC Transceiver | A24A0MR8 | 1 |
| Q-Free ® ACC650 | RSE650 Connection Box | A1400002 | 1 |
| Q-Free ® ACC652 | RSE650 Generic Bracket (round tube, 2 angles, w/ ACC650 holder) | B11BGMR8 | 1 |
| Planet POE-161 | Planet Single POE Injector | A3CE0006 | 1 |
|  | CAT5 Cable (5m) |  | 1 |
|  | Mini Inline Coupler |  | 1 |
| Q-Free ® RSE622 | Handheld DSRC Transceiver | A24A0MRB | 1 |

## 2.2 Software

Table 2 shows the software used included in the kit.

Table 2: Software

| Name | Description | Part Number | Count |
|---|---|---|---|
| MDMRAPP SW RSE650 | Multipurpose application for RSE650 | A9E65003 | 1 |

## 2.3 Tags

Table 3 shows the DSRC tags included in the kit.

Table 3: Tags

| Part Number | Description | Customer Product Specification | Count |
|---|---|---|---|
| A2Rxxxx1 | OBU615 w/ static RTM data | a2rxxxx1_tachograph.xml | 1 |
| A2Rxxxx2 | OBU615 w/ static RTM data | a2rxxxx2_tachograph.xml | 1 |

Table 3: (continued)

| Part Number | Description | Customer Product Specification | Count |
|---|---|---|---|
| A2Rxxxx3 | OBU615 w/ static RTM data | a2rxxxx3_tachograph.xml | 1 |
| A2R7AC01 | OBU611 w/ dynamic RTM data | a2r7ac01_tachograph.xml | 1 |

# 3 QF Packages Installation

Table 4 shows the additional packages that have been installed on your RSE650.

Table 4: QF Packages

| Name | Version | Revision |
|---|---|---|
| qfree-rse650-libqa1c | 1.0.0-04 | 80269 |
| qfree-rse650-libdsrcl2 | 1.1.0-04 | 80271 |
| qfree-rse650-app-mdmrapp | 7.01-09 | 80714 |

See [1] section 5.1.4 for how to check that the correct packages have been installed on your RSE650.

# 4 RSE650 Configuration

Your RSE650 has been pre-configured as a single-gantry reader.

## 4.1 Change IP Address

The default IP address of the RSE650 is *192.168.127.81*. See [1] section 5.1.1 for how to change the IP address.

**Note**
You must be on the same subnet (e.g. *192.168.127.xyz*) as the reader to be able to reach the reader.

## 4.2 Advanced Test Configuration

See tip box on the bottom of page 5 and section 10 in [2] for tips on how to decrease time between reads of the same OBU in a test scenario.

# 5 Transaction Model Installation

# 6 RSE650 Link Testing

See [1] section 5.1.3 for how to perform link testing.

# 7 RSE650 Transaction Logging

See [2] section 2.1 for how to access the transaction log. Clicking on the *xml* link under the *Layer7XmlLog* column allows you to see the content of the OBU transaction.

See [2] sections 2.2, 3 and 4.1-4.3 for how to implement logging to an external HTTP server.

# 6. Annex 2 - Manual of use of the development kit

# Q-Free Tachograph Software Development Kit

Version 1.0.0

# Contents

# Chapter 1

# Q-Free Tachograph Software Development Kit

This software development kit (SDK) consists of a library and an example application.

The library provides an interface for reading and writing the RTM data element in an OBU611 tag.

The example application demonstrates the following:

- Communication with OBU611 (QFTOP over UART)
  1. Open serial port
  2. Write QFTOP message(s) to serial port
  3. Read QFTOP messages(s) from serial port
- Tachograph Client
  1. Read RTM data
  2. Write RTM data

## 1.1 Platform Support

Both x86 and x64 is supported.

## 1.2 Tachograph Library Dependencies

Compiler:

- g++-4.9
- clang-3.7

Compile Dependencies:

- libboost1.55-all-dev
- g++-4.9-multilib (if compiling x64 binaries from i386)

## 1.3 Build Instructions

### 1.3.1 Example Tachograph Application

g++:

```
./gradlew clean tachographAppX64ReleaseExecutable
```

clang:

```
./gradlew clean tachographAppX64ReleaseExecutable -PuseClang
```

### 1.3.2 Tachograph Library

g++:

```
./gradlew clean tachographLibX64ReleaseExecutable
```

clang:

```
./gradlew clean tachographLibX64ReleaseExecutable -PuseClang
```

# Chapter 2

# Module Index

## 2.1 Modules

Here is a list of all modules:

# Chapter 3

# Namespace Index

## 3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all files with brief descriptions:

# Chapter 6

# Module Documentation

## 6.1 Tachograph

### Classes

- class tachograph::application< TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH >

  *Tachograph Client.*

### 6.1.1 Detailed Description

Elements related to the tachograph client.

## 6.2 QFTOP

### Classes

- struct qftop::set_control

  *Set Control Data.*
- struct qftop::attribute_list

  *Attribute List.*
- struct qftop::write_without_cred

  *Write Without Credentials Request.*
- struct qftop::get_control

  *Get Control Data.*
- struct qftop::read_without_cred

  *Read Without Credentials Request.*
- struct qftop::read_without_cred_response

  *Read Without Credentials Response.*
- class qftop::write_response_callback

  *Write Response Callback Interface.*
- class qftop::read_response_callback

  *Read Response Callback Interface.*
- class qftop::application

  *QFTOP Client.*
- struct qfTopMSG

  *Structure of QFTop Message.*
- struct qfTopMessage

  *State holder of QFTop Message plus state info when parsing.*

### Macros

- #define MAXIMUMQFTOPFRAMESIZE 200

### Enumerations

- enum qftop_cmd_type_t {
  QFTOP_ECHO_REQ = 0x00, QFTOP_ECHO_RESP = 0x80, QFTOP_ACK = 0x01, QFTOP_NACK = 0x02,
  QFTOP_MMI_REQ = 0x30, QFTOP_INIT_NOTIFICATION = 0x31, QFTOP_TRANSP_RESP = 0x33, QFTOP_TRANSP_REQ = 0x34,
  QFTOP_REGISTER_APP_REQ = 0x36, QFTOP_REGISTER_APP = 0x37, QFTOP_TEST_REQ = 0x38, QFTOP_TEST_RESP = 0x39,
  QFTOP_PERS_REQ = 0x3A, QFTOP_PERS_RESP = 0x3B, QFTOP_DSRC_L7_REQ = 0x3C, QFTOP_DSRC_L7_RESP = 0x3D,
  QFTOP_TRACE_LOG_REQ = 0xF0, QFTOP_TRACE_LOG_RESP = 0xF1 }

  *Types of QFTOP messages.*
- enum qftop_Types {
  Application = 0, ACK = 1, NACK = 2, dsrc_l7_req = 0x3C,
  dsrc_l7_resp = 0x3D, crc_init = 0x6363, qftop_preamble = 0xB5, maximumQFTOPFrameSize = MAXIMUMQFTOPFRAMESIZE }

  *Types of QFTOP messages.*

### Functions

- void qftop::print_message (std::ostream &out, const qfTopMessage ∗rhs)

  *Print QFTOP message to stream.*
- std::ostream & qftop::operator<< (std::ostream &out, const set_control &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const attribute_list &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const write_without_cred &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const get_control &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const read_without_cred &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const read_without_cred_response &rhs)
- int qftop_parse (struct qfTopMessage ∗msg, uint8_t cr)

  *Function to parse a new byte into a message being received.*
- uint16_t qftop_extractMessage (struct qfTopMessage ∗msg_out, struct qfTopMessage ∗msg_in)

  *Function to build an internal message based on bytes in another message.*
- void qftop_addToCRC (struct qfTopMessage ∗msg, uint8_t ch)

  *Modify crc calculation for a new byte.*
- void qftop_addParameter (struct qfTopMessage ∗msg, uint8_t p)

  *Function for adding a single parameter to a QFTop message.*
- void qftop_clear (struct qfTopMessage ∗msg)

  *Convenience function to zero / reset a message.*
- void qftop_application (struct qfTopMessage ∗msg)

  *Convenience function to initialise an application message (dsrc req)*
- void qftop_ack (struct qfTopMessage ∗msg)

  *Convenience function to build an ACK message.*
- unsigned int qftop_buildMessage (struct qfTopMessage ∗msg)

  *Function to build a byte stream ready for transmission based on a message.*

## 6.2.1 Detailed Description

Elements related to QFTOP

## 6.2.2 Macro Definition Documentation

### 6.2.2.1 #define MAXIMUMQFTOPFRAMESIZE 200

Definition at line 12 of file qftop_client.h.

## 6.2.3 Enumeration Type Documentation

### 6.2.3.1 enum qftop_cmd_type_t

Types of QFTOP messages.

**Enumerator**

***QFTOP_ECHO_REQ***

***QFTOP_ECHO_RESP***

***QFTOP_ACK***

***QFTOP_NACK***

***QFTOP_MMI_REQ***

***QFTOP_INIT_NOTIFICATION***

***QFTOP_TRANSP_RESP***

***QFTOP_TRANSP_REQ***

***QFTOP_REGISTER_APP_REQ***

***QFTOP_REGISTER_APP***

***QFTOP_TEST_REQ***

***QFTOP_TEST_RESP***

***QFTOP_PERS_REQ***

***QFTOP_PERS_RESP***

***QFTOP_DSRC_L7_REQ***

***QFTOP_DSRC_L7_RESP***

***QFTOP_TRACE_LOG_REQ***

***QFTOP_TRACE_LOG_RESP***

Definition at line 19 of file qftop_client.h.

#### 6.2.3.2 enum qftop_Types

Types of QFTOP messages.

**Enumerator**

***Application***

***ACK***

***NACK***

***dsrc_l7_req***

***dsrc_l7_resp***

***crc_init***

***qftop_preamble***

***maximumQFTOPFrameSize***

Definition at line 42 of file qftop_client.h.

### 6.2.4 Function Documentation

#### 6.2.4.1 std::ostream & qftop::operator<< ( std::ostream & *out*, const set_control & *rhs* )

Definition at line 22 of file qftop_application.cpp.

#### 6.2.4.2 std::ostream & qftop::operator<< ( std::ostream & *out*, const attribute_list & *rhs* )

Definition at line 30 of file qftop_application.cpp.

#### 6.2.4.3 std::ostream & qftop::operator<< ( std::ostream & *out*, const write_without_cred & *rhs* )

Definition at line 41 of file qftop_application.cpp.

#### 6.2.4.4 std::ostream & qftop::operator<< ( std::ostream & *out*, const get_control & *rhs* )

Definition at line 50 of file qftop_application.cpp.

#### 6.2.4.5 std::ostream & qftop::operator<< ( std::ostream & *out*, const read_without_cred & *rhs* )

Definition at line 58 of file qftop_application.cpp.

#### 6.2.4.6 std::ostream & qftop::operator<< ( std::ostream & *out*, const read_without_cred_response & *rhs* )

Definition at line 67 of file qftop_application.cpp.

#### 6.2.4.7 void qftop::print_message ( std::ostream & *out*, const **qfTopMessage** ∗ *rhs* )

Print QFTOP message to stream.

**Parameters**

| | | |
|---|---|---|
| in | *out* | output stream |
| in | *rhs* | QFTOP message to print |

Definition at line 6 of file qftop_application.cpp.

#### 6.2.4.8 void qftop_ack ( struct **qfTopMessage** ∗ *msg* )

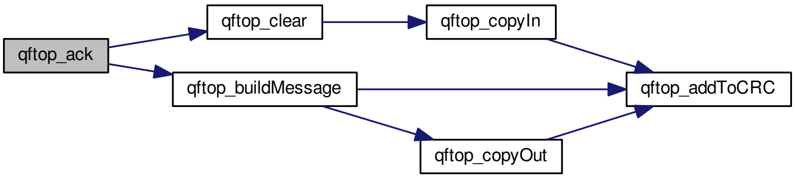
Convenience function to build an ACK message.

**Parameters**

| | | |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |

Definition at line 109 of file qftop_client.c.

Here is the call graph for this function:



#### 6.2.4.9 void qftop_addParameter ( struct **qfTopMessage** ∗ *msg*, uint8_t *p* )

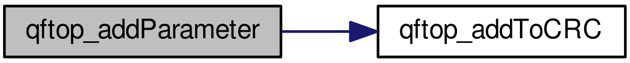
Function for adding a single parameter to a QFTop message.

**Parameters**

| | | |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |
| in | *p* | Parameter byte being added |

Definition at line 133 of file qftop_client.c.

Here is the call graph for this function:



#### 6.2.4.10 void qftop_addToCRC ( struct **qfTopMessage** ∗ *msg,* uint8_t *ch* )

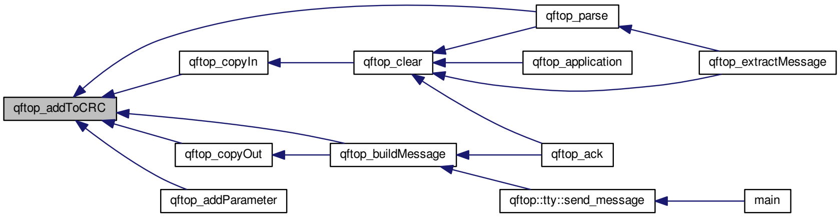
Modify crc calculation for a new byte.

**Parameters**

| | | |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |
| in | *ch* | byte being added |

Definition at line 8 of file qftop_client.c.

Here is the caller graph for this function:



#### 6.2.4.11 void qftop_application ( struct **qfTopMessage** ∗ *msg* )

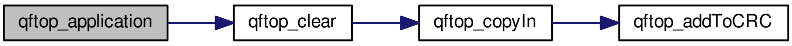
Convenience function to initialise an application message (dsrc req)

**Parameters**

| | | |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |

Definition at line 121 of file qftop_client.c.

Here is the call graph for this function:



#### 6.2.4.12 unsigned int qftop_buildMessage ( struct **qfTopMessage** ∗ *msg* )

Function to build a byte stream ready for transmission based on a message.

**Parameters**

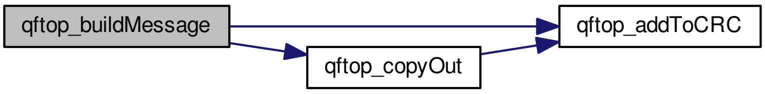
|  |  |  |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |

**Returns**

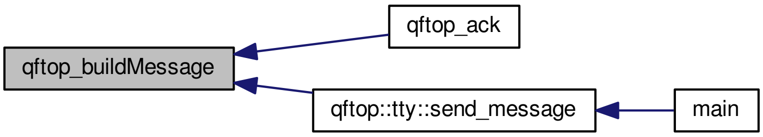
len final length of message

Definition at line 81 of file qftop_client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.2.4.13 void qftop_clear ( struct **qfTopMessage** ∗ *msg* )

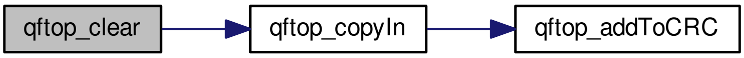
Convenience function to zero / reset a message.

**Parameters**

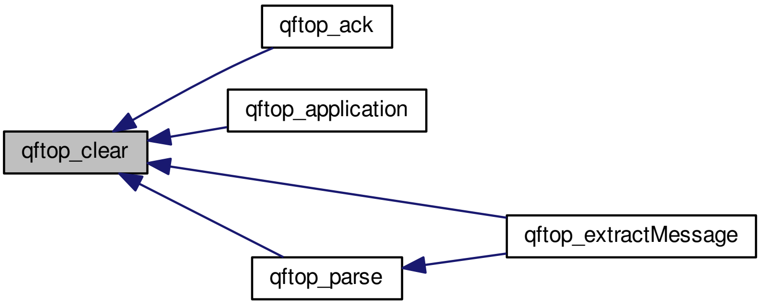
|  |  |  |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |

Definition at line 68 of file qftop_client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.2.4.14 uint16_t qftop_extractMessage ( struct **qfTopMessage** ∗ *msg_out,* struct **qfTopMessage** ∗ *msg_in* )

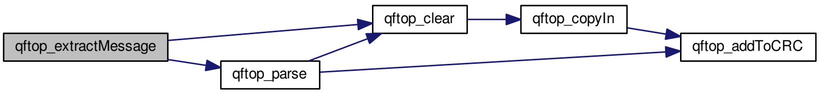
Function to build an internal message based on bytes in another message.

**Parameters**

| | | |
|---|---|---|
| in | *msg_in* | pointer to qfTopMessage |
| out | *msg_out* | pointer to qfTopMessage |

Definition at line 138 of file qftop_client.c.

Here is the call graph for this function:



#### 6.2.4.15 int qftop_parse ( struct **qfTopMessage** ∗ *msg,* uint8_t *cr* )

Function to parse a new byte into a message being received.

This function is pretty ruthless. It will continue passing bytes till a message is complete or buffer overflow. If there's a protocol error the message will be restarted.

**Parameters**

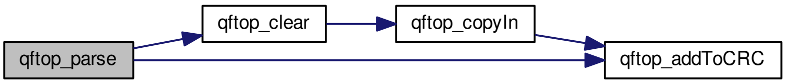
| | | |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |
| in | *cr* | new byte |

**Returns**

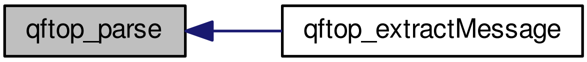
> 0 once a complete message is received.
= 0 if message is not yet complete
< 0 on buffer overflow

Definition at line 150 of file qftop_client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.3 Serial Port

QFTOP over Serial Port.

QFTOP over Serial Port. Elements related to serial port communication.

## 6.4 Utility

String Helper Functions.

### Namespaces

- string_extra

### 6.4.1 Detailed Description

String Helper Functions. Convenience methods.

## 6.5 Example

Example Tachograph Application.

### Functions

- int main ()

### 6.5.1 Detailed Description

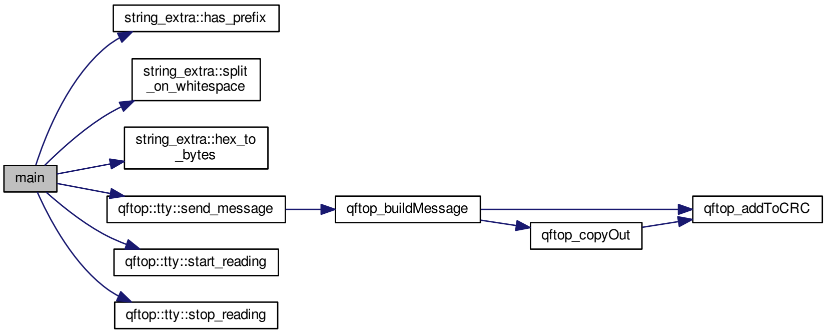
Example Tachograph Application. Example Tachograph Application

### 6.5.2 Function Documentation

#### 6.5.2.1 int main ( )

Definition at line 22 of file main.cpp.

Here is the call graph for this function:



# Chapter 7

# Namespace Documentation

## 7.1 qftop Namespace Reference

### Classes

- struct set_control

  *Set Control Data.*
- struct attribute_list

  *Attribute List.*
- struct write_without_cred

  *Write Without Credentials Request.*
- struct get_control

  *Get Control Data.*
- struct read_without_cred

  *Read Without Credentials Request.*
- struct read_without_cred_response

  *Read Without Credentials Response.*
- class write_response_callback

  *Write Response Callback Interface.*
- class read_response_callback

  *Read Response Callback Interface.*
- class application

  *QFTOP Client.*
- class tty

### Functions

- void print_message (std::ostream &out, const qfTopMessage ∗rhs)

  *Print QFTOP message to stream.*
- std::ostream & operator<< (std::ostream &out, const set_control &rhs)
- std::ostream & operator<< (std::ostream &out, const attribute_list &rhs)
- std::ostream & operator<< (std::ostream &out, const write_without_cred &rhs)
- std::ostream & operator<< (std::ostream &out, const get_control &rhs)
- std::ostream & operator<< (std::ostream &out, const read_without_cred &rhs)
- std::ostream & operator<< (std::ostream &out, const read_without_cred_response &rhs)

## 7.2 string_extra Namespace Reference

### Functions

- bool has_prefix (std::string string_to_check, std::string prefix)

  *Check if string has prefix.*
- std::vector< std::string > split_on_whitespace (std::string string_to_split)

  *Split string on whitespace.*
- std::vector< unsigned char > hex_to_bytes (std::string hex_string)

  *Convert hex string to bytes.*

### 7.2.1 Function Documentation

#### 7.2.1.1 bool string_extra::has_prefix ( std::string *string_to_check,* std::string *prefix* )

Check if string has prefix.

**Parameters**

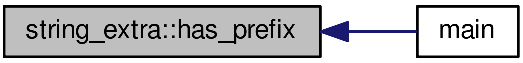
| | | |
|---|---|---|
| in | *string_to_check* | string that is checked for prefix |
| in | *prefix* | the prefix that is checked for |

**Returns**

true if string has prefix, otherwise false

Definition at line 6 of file string_extra.cpp.

Here is the caller graph for this function:



#### 7.2.1.2 std::vector< unsigned char > string_extra::hex_to_bytes ( std::string *hex_string* )

Convert hex string to bytes.

**Parameters**

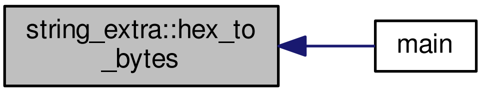
| | | |
|---|---|---|
| in | *hex_string* | hex string |

**Returns**

bytes

Definition at line 20 of file string_extra.cpp.

Here is the caller graph for this function:



#### 7.2.1.3 std::vector< std::string > string_extra::split_on_whitespace ( std::string *string_to_split* )

Split string on whitespace.

**Parameters**

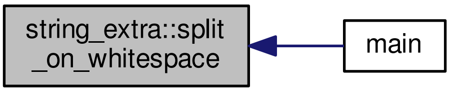
| | | |
|---|---|---|
| `in` | *string_to_split* | string to be split |

**Returns**

the split string

Definition at line 10 of file string_extra.cpp.

Here is the caller graph for this function:



## 7.3 tachograph Namespace Reference

### Classes

- class application

  *Tachograph Client.*

# Chapter 8

# Class Documentation

## 8.1 tachograph::application< TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH > Class Template Reference

Tachograph Client.

```
#include <tachograph.hpp>
```

### Public Member Functions

- application (std::shared_ptr< qftop::application > qftop_application_ptr, std::shared_ptr< std::ostream > output_stream_ptr)

  *Constructor.*
- void read_rtm_data (std::function< void(std::array< unsigned char, TACHOGRAPH_PAYLOAD_LENGTH > tachograph_payload, std::array< unsigned char, DSRC_SECURITY_DATA_LENGTH > dsrc_security_data)> callback)

  *read RTM data without credentials*
- void write_rtm_data (std::array< unsigned char, TACHOGRAPH_PAYLOAD_LENGTH > tachograph_payload, std::array< unsigned char, DSRC_SECURITY_DATA_LENGTH > dsrc_security_data)

  *write RTM data without credentials*

### 8.1.1 Detailed Description

**template<std::size_t TACHOGRAPH_PAYLOAD_LENGTH, std::size_t DSRC_SECURITY_DATA_LENGTH>class tachograph::application< TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH >**

Tachograph Client.

**Template Parameters**

| | |
|---|---|
| *TACHOGRAPH_PAYLOAD_LENGTH* | length of tachograph payload |
| *DSRC_SECURITY_DATA_LENGTH* | length of DSRC security data |

Definition at line 26 of file tachograph.hpp.

### 8.1.2 Constructor & Destructor Documentation

#### 8.1.2.1 template< std::size_t TACHOGRAPH_PAYLOAD_LENGTH, std::size_t DSRC_SECURITY_DATA_LENGTH> **tachograph::application**< TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH >**::application** ( std::shared_ptr< **qftop::application**< TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH > > *qftop_application_ptr,* std::shared_ptr< std::ostream > *output_stream_ptr* )

Constructor.

Definition at line 8 of file tachograph.ipp.

### 8.1.3 Member Function Documentation

#### 8.1.3.1 template< std::size_t TACHOGRAPH_PAYLOAD_LENGTH, std::size_t DSRC_SECURITY_DATA_LENGTH> void **tachograph::application**< TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH >::read_rtm_data ( std::function< void(std::array< unsigned char, TACHOGRAPH_PAYLOAD_LENGTH > tachograph_payload, std::array< unsigned char, DSRC_SECURITY_DATA_LENGTH > dsrc_security_data)> *callback* )

read RTM data without credentials

**Parameters**

| | | |
|---|---|---|
| `in` | *callback* | callback called when RTM data message is received |

Definition at line 14 of file tachograph.ipp.

#### 8.1.3.2 template< std::size_t TACHOGRAPH_PAYLOAD_LENGTH, std::size_t DSRC_SECURITY_DATA_LENGTH> void **tachograph::application**< TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH >::write_rtm_data ( std::array< unsigned char, TACHOGRAPH_PAYLOAD_LENGTH > *tachograph_payload,* std::array< unsigned char, DSRC_SECURITY_DATA_LENGTH > *dsrc_security_data* )

write RTM data without credentials

**Parameters**

| | | |
|---|---|---|
| `in` | *tachograph_-payload* | content of tachograph payload to be written |
| `in` | *dsrc_security_-data* | content of DSRC security data to be written |

Definition at line 52 of file tachograph.ipp.

## 8.2 qftop::application Class Reference

QFTOP Client.

```
#include <qftop_application.hpp>
```

### Public Member Functions

- application (std::function< void(std::unique_ptr< qfTopMessage >)> on_message_write_callback, std::shared_ptr< std::ostream > output_stream)

  *Constructor.*
- void push_message (std::unique_ptr< qfTopMessage > message)

  *Add message to back of input buffer.*
- void start_polling ()

  *Start polling for messages.*
- void stop_polling ()

  *Stop polling for messages.*

- void send_write_without_cred (unsigned char element_id, unsigned char attribute_id, std::vector< unsigned char > attribute_value, std::shared_ptr< write_response_callback > on_write_response_callback)

  *write a single attribute without credentials*
- void send_read_without_cred (unsigned char element_id, unsigned char attribute_id, std::shared_ptr< read_response_callback > on_read_response_callback)

  *read a single attribute without credentials*

### 8.2.1 Detailed Description

QFTOP Client.

Definition at line 185 of file qftop_application.hpp.

### 8.2.2 Constructor & Destructor Documentation

#### 8.2.2.1 qftop::application::application ( std::function< void(std::unique_ptr< **qfTopMessage** >)> *on_message_write_callback,* std::shared_ptr< std::ostream > *output_stream* )

Constructor.

Definition at line 83 of file qftop_application.cpp.

### 8.2.3 Member Function Documentation

#### 8.2.3.1 void qftop::application::push_message ( std::unique_ptr< **qfTopMessage** > *message* )

Add message to back of input buffer.

**Parameters**

| | | |
|---|---|---|
| `in` | *message* | message to add to back of input buffer |

Definition at line 92 of file qftop_application.cpp.

#### 8.2.3.2 void qftop::application::send_read_without_cred ( unsigned char *element_id,* unsigned char *attribute_id,* std::shared_ptr< **read_response_callback** > *on_read_response_callback* )

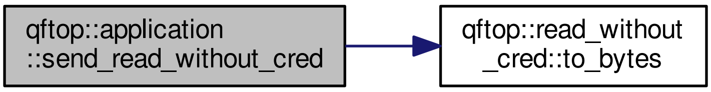
read a single attribute without credentials

**Parameters**

| | | |
|---|---|---|
| `in` | *element_id* | ID of the element containing the attribute to be read |
| `in` | *attribute_id* | ID of the attribute to be read |
| `in` | *on_read_response_callback* | callback called when the read response is received |

Definition at line 184 of file qftop_application.cpp.

Here is the call graph for this function:


qftop::application ::send_read_without_cred
qftop::read_without _cred::to_bytes

#### 8.2.3.3 void qftop::application::send_write_without_cred ( unsigned char *element_id,* unsigned char *attribute_id,* std::vector< unsigned char > *attribute_value,* std::shared_ptr< **write_response_callback** > *on_write_response_callback* )

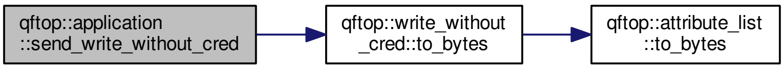
write a single attribute without credentials

**Parameters**

| | | |
|---|---|---|
| in | *element_id* | ID of element containing the attribute to be written |
| in | *attribute_id* | ID of attribute to be written to |
| in | *attribute_value* | attribute value to be written |
| in | *on_write_response_callback* | callback called when the write response is received |

Definition at line 151 of file qftop_application.cpp.

Here is the call graph for this function:


qftop::application ::send_write_without_cred
qftop::write_without _cred::to_bytes
qftop::attribute_list ::to_bytes

#### 8.2.3.4 void qftop::application::start_polling ( )

Start polling for messages.

Definition at line 132 of file qftop_application.cpp.

#### 8.2.3.5 void qftop::application::stop_polling ( )

Stop polling for messages.

Definition at line 142 of file qftop_application.cpp.

## 8.3 qftop::attribute_list Struct Reference

Attribute List.

```
#include <qftop_application.hpp>
```

## Public Member Functions

- std::vector< unsigned char > to_bytes ()

## Public Attributes

- unsigned char attribute_id
- unsigned char container_type
- std::vector< unsigned char > attribute_value

## Friends

- std::ostream & operator<< (std::ostream &, const attribute_list &)

### 8.3.1 Detailed Description

Attribute List.

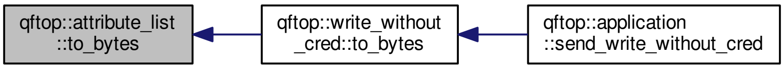
Definition at line 53 of file qftop_application.hpp.

### 8.3.2 Member Function Documentation

#### 8.3.2.1 std::vector<unsigned char> qftop::attribute_list::to_bytes ( )

Definition at line 57 of file qftop_application.hpp.

Here is the caller graph for this function:


qftop::attribute_list ::to_bytes
qftop::write_without _cred::to_bytes
qftop::application ::send_write_without_cred

### 8.3.3 Friends And Related Function Documentation

#### 8.3.3.1 std::ostream& operator<< ( std::ostream & , const attribute_list & ) [friend]

Definition at line 30 of file qftop_application.cpp.

### 8.3.4 Member Data Documentation

#### 8.3.4.1 unsigned char qftop::attribute_list::attribute_id

Definition at line 54 of file qftop_application.hpp.

#### 8.3.4.2 std::vector<unsigned char> qftop::attribute_list::attribute_value

Definition at line 56 of file qftop_application.hpp.

#### 8.3.4.3 unsigned char qftop::attribute_list::container_type

Definition at line 55 of file qftop_application.hpp.

## 8.4 qfTopMSG::crc Union Reference

```
#include <qftop_client.h>
```

### Public Attributes

- uint8_t bytes [2]
- uint16_t word

### 8.4.1 Detailed Description

Definition at line 76 of file qftop_client.h.

### 8.4.2 Member Data Documentation

#### 8.4.2.1 uint8_t qfTopMSG::crc::bytes[2]

< check sums

Definition at line 77 of file qftop_client.h.

#### 8.4.2.2 uint16_t qfTopMSG::crc::word

Definition at line 78 of file qftop_client.h.

## 8.5 qftop::get_control Struct Reference

Get Control Data.

```
#include <qftop_application.hpp>
```

### Public Attributes

- union {
    struct {
      unsigned char mode: 1
      unsigned char has_attribute_list: 1
      unsigned char has_iid: 1
      unsigned char has_credentials: 1
      unsigned char action: 4
    }
    unsigned char byte
  };

## Friends

- std::ostream & operator<< (std::ostream &, const get_control &)

### 8.5.1 Detailed Description

Get Control Data.

Definition at line 98 of file qftop_application.hpp.

### 8.5.2 Friends And Related Function Documentation

#### 8.5.2.1 std::ostream& operator<< ( std::ostream & , const get_control & ) [friend]

Definition at line 50 of file qftop_application.cpp.

### 8.5.3 Member Data Documentation

#### 8.5.3.1 union { ... }

#### 8.5.3.2 unsigned char qftop::get_control::action

Definition at line 105 of file qftop_application.hpp.

#### 8.5.3.3 unsigned char qftop::get_control::byte

Definition at line 107 of file qftop_application.hpp.

#### 8.5.3.4 unsigned char qftop::get_control::has_attribute_list

Definition at line 102 of file qftop_application.hpp.

#### 8.5.3.5 unsigned char qftop::get_control::has_credentials

Definition at line 104 of file qftop_application.hpp.

#### 8.5.3.6 unsigned char qftop::get_control::has_iid

Definition at line 103 of file qftop_application.hpp.

#### 8.5.3.7 unsigned char qftop::get_control::mode

Definition at line 101 of file qftop_application.hpp.

## 8.6 qfTopMSG::header Union Reference

```
#include <qftop_client.h>
```

## Public Attributes

- struct {
  unsigned char preamble: 8
  unsigned char sequence: 4
  unsigned char frameType: 2
  unsigned char status: 1
  unsigned char syn: 1
  unsigned char length: 8
  };
- uint8_t bytes [3]

### 8.6.1 Detailed Description

Definition at line 58 of file qftop_client.h.

### 8.6.2 Member Data Documentation

#### 8.6.2.1 struct { ... }

#### 8.6.2.2 uint8_t qfTopMSG::header::bytes[3]

Definition at line 67 of file qftop_client.h.

#### 8.6.2.3 unsigned char qfTopMSG::header::frameType

Definition at line 62 of file qftop_client.h.

#### 8.6.2.4 unsigned char qfTopMSG::header::length

Definition at line 65 of file qftop_client.h.

#### 8.6.2.5 unsigned char qfTopMSG::header::preamble

< QFTop header

Definition at line 60 of file qftop_client.h.

#### 8.6.2.6 unsigned char qfTopMSG::header::sequence

Definition at line 61 of file qftop_client.h.

#### 8.6.2.7 unsigned char qfTopMSG::header::status

Definition at line 63 of file qftop_client.h.

#### 8.6.2.8 unsigned char qfTopMSG::header::syn

Definition at line 64 of file qftop_client.h.

## 8.7 qfTopMSG::pdu Union Reference

```
#include <qftop_client.h>
```

### Public Attributes

- struct {
  uint8_t messageType
  uint8_t PARAMETERS [maximumQFTOPFrameSize]
  };
- uint8_t bytes [maximumQFTOPFrameSize+1]

### 8.7.1 Detailed Description

Definition at line 69 of file qftop_client.h.

### 8.7.2 Member Data Documentation

#### 8.7.2.1 struct { ... }

< The ASN.1 or other data payload

#### 8.7.2.2 uint8_t qfTopMSG::pdu::bytes[maximumQFTOPFrameSize+1]

Definition at line 74 of file qftop_client.h.

#### 8.7.2.3 uint8_t qfTopMSG::pdu::messageType

Definition at line 71 of file qftop_client.h.

#### 8.7.2.4 uint8_t qfTopMSG::pdu::PARAMETERS[maximumQFTOPFrameSize]

Definition at line 72 of file qftop_client.h.

## 8.8 qfTopMessage Struct Reference

State holder of QFTop Message plus state info when parsing.

```
#include <qftop_client.h>
```

### Public Attributes

- struct qfTopMSG msg
- bool pre_escape
- bool head_start
- bool head_read
- uint8_t message [maximumQFTOPFrameSize+10]
- unsigned int message_length

### 8.8.1 Detailed Description

State holder of QFTop Message plus state info when parsing.

Definition at line 85 of file qftop_client.h.

### 8.8.2 Member Data Documentation

#### 8.8.2.1 bool qfTopMessage::head_read

state: header read

Definition at line 89 of file qftop_client.h.

#### 8.8.2.2 bool qfTopMessage::head_start

state: reading header

Definition at line 88 of file qftop_client.h.

#### 8.8.2.3 uint8_t qfTopMessage::message[maximumQFTOPFrameSize+10]

serialised message

Definition at line 90 of file qftop_client.h.

#### 8.8.2.4 unsigned int qfTopMessage::message_length

length of serialised message

Definition at line 91 of file qftop_client.h.

#### 8.8.2.5 struct qfTopMSG qfTopMessage::msg

The message being sent/received

Definition at line 86 of file qftop_client.h.

#### 8.8.2.6 bool qfTopMessage::pre_escape

state: preamble must be escaped

Definition at line 87 of file qftop_client.h.

## 8.9 qfTopMSG Struct Reference

Structure of QFTop Message.

```
#include <qftop_client.h>
```

### Classes

- union crc
- union header
- union pdu

## Public Attributes

- union qfTopMSG::header HEADER
- union qfTopMSG::pdu PDU
- union qfTopMSG::crc CRC
- union qfTopMSG::crc CRC_REC

### 8.9.1 Detailed Description

Structure of QFTop Message.

see the specification doc for details.

Definition at line 57 of file qftop_client.h.

### 8.9.2 Member Data Documentation

#### 8.9.2.1 union **qfTopMSG::crc** qfTopMSG::CRC

generated from passing

#### 8.9.2.2 union **qfTopMSG::crc** qfTopMSG::CRC_REC

received

#### 8.9.2.3 union **qfTopMSG::header** qfTopMSG::HEADER

#### 8.9.2.4 union **qfTopMSG::pdu** qfTopMSG::PDU

## 8.10 qftop::read_response_callback Class Reference

Read Response Callback Interface.

`#include <qftop_application.hpp>`

## Public Member Functions

- virtual void on_success (std::vector< unsigned char > attribute_value)=0
- virtual void on_error ()=0

### 8.10.1 Detailed Description

Read Response Callback Interface.

Definition at line 176 of file qftop_application.hpp.

### 8.10.2 Member Function Documentation

#### 8.10.2.1 virtual void qftop::read_response_callback::on_error ( ) `[pure virtual]`

#### 8.10.2.2 virtual void qftop::read_response_callback::on_success ( std::vector< unsigned char > *attribute_value* ) `[pure virtual]`

## 8.11 qftop::read_without_cred Struct Reference

Read Without Credentials Request.

`#include <qftop_application.hpp>`

### Public Member Functions

- std::vector< unsigned char > to_bytes ()

### Public Attributes

```
• union {
    struct {
      unsigned char length
      unsigned char fragment_header
      struct get_control control
      unsigned char element_id
      unsigned char attribute_count
      unsigned char attribute_list [1]
    }
    unsigned char bytes [6]
  };
```

### Friends

- std::ostream & operator<< (std::ostream &, const read_without_cred &)

### 8.11.1 Detailed Description

Read Without Credentials Request.

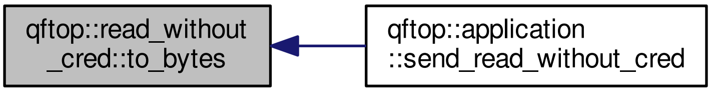
Definition at line 117 of file qftop_application.hpp.

### 8.11.2 Member Function Documentation

#### 8.11.2.1 std::vector<unsigned char> qftop::read_without_cred::to_bytes ( )

Definition at line 129 of file qftop_application.hpp.

Here is the caller graph for this function:


qftop::read_without_cred::to_bytes
qftop::application::send_read_without_cred

### 8.11.3 Friends And Related Function Documentation

#### 8.11.3.1 std::ostream& operator<< ( std::ostream & , const read_without_cred & ) [friend]

Definition at line 58 of file qftop_application.cpp.

### 8.11.4 Member Data Documentation

#### 8.11.4.1 union { ... }

#### 8.11.4.2 unsigned char qftop::read_without_cred::attribute_count

Definition at line 124 of file qftop_application.hpp.

#### 8.11.4.3 unsigned char qftop::read_without_cred::attribute_list[1]

Definition at line 125 of file qftop_application.hpp.

#### 8.11.4.4 unsigned char qftop::read_without_cred::bytes[6]

Definition at line 127 of file qftop_application.hpp.

#### 8.11.4.5 struct get_control qftop::read_without_cred::control

Definition at line 122 of file qftop_application.hpp.

#### 8.11.4.6 unsigned char qftop::read_without_cred::element_id

Definition at line 123 of file qftop_application.hpp.

#### 8.11.4.7 unsigned char qftop::read_without_cred::fragment_header

Definition at line 121 of file qftop_application.hpp.

#### 8.11.4.8 unsigned char qftop::read_without_cred::length

Definition at line 120 of file qftop_application.hpp.

## 8.12 qftop::read_without_cred_response Struct Reference

Read Without Credentials Response.

`#include <qftop_application.hpp>`

### Public Attributes

- union {
  struct {
  unsigned char length
  unsigned char fragment_header

```
        struct get_control control
        unsigned char element_id
        unsigned char attribute_count
        unsigned char attribute_id
        unsigned char container_id
      }
      unsigned char bytes [sizeof(unsigned char)∗6+sizeof(get_control)]
    } header
```

- std::vector< unsigned char > attribute_value

## Friends

- std::ostream & operator<< (std::ostream &, const read_without_cred_response &)

### 8.12.1 Detailed Description

Read Without Credentials Response.

Definition at line 145 of file qftop_application.hpp.

### 8.12.2 Friends And Related Function Documentation

#### 8.12.2.1 std::ostream& operator<< ( std::ostream & , const read_without_cred_response & ) [friend]

Definition at line 67 of file qftop_application.cpp.

### 8.12.3 Member Data Documentation

#### 8.12.3.1 unsigned char qftop::read_without_cred_response::attribute_count

Definition at line 152 of file qftop_application.hpp.

#### 8.12.3.2 unsigned char qftop::read_without_cred_response::attribute_id

Definition at line 153 of file qftop_application.hpp.

#### 8.12.3.3 std::vector<unsigned char> qftop::read_without_cred_response::attribute_value

Definition at line 158 of file qftop_application.hpp.

#### 8.12.3.4 unsigned char qftop::read_without_cred_response::bytes[sizeof(unsigned char)∗6+sizeof(get_control)]

Definition at line 156 of file qftop_application.hpp.

#### 8.12.3.5 unsigned char qftop::read_without_cred_response::container_id

Definition at line 154 of file qftop_application.hpp.

#### 8.12.3.6 struct get_control qftop::read_without_cred_response::control

Definition at line 150 of file qftop_application.hpp.

#### 8.12.3.7 unsigned char qftop::read_without_cred_response::element_id

Definition at line 151 of file qftop_application.hpp.

#### 8.12.3.8 unsigned char qftop::read_without_cred_response::fragment_header

Definition at line 149 of file qftop_application.hpp.

#### 8.12.3.9 union { ... } qftop::read_without_cred_response::header

#### 8.12.3.10 unsigned char qftop::read_without_cred_response::length

Definition at line 148 of file qftop_application.hpp.

## 8.13 qftop::set_control Struct Reference

Set Control Data.

```
#include <qftop_application.hpp>
```

### Public Attributes

- union {
    struct {
      unsigned char mode: 1
      unsigned char fill: 1
      unsigned char has_iid: 1
      unsigned char has_credentials: 1
      unsigned char action: 4
    }
    unsigned char byte
  };

### Friends

- std::ostream & operator<< (std::ostream &, const set_control &)

### 8.13.1 Detailed Description

Set Control Data.

Definition at line 34 of file qftop_application.hpp.

### 8.13.2 Friends And Related Function Documentation

#### 8.13.2.1 std::ostream& operator<< ( std::ostream & , const set_control & ) [friend]

Definition at line 22 of file qftop_application.cpp.

### 8.13.3 Member Data Documentation

#### 8.13.3.1 union { ... }

#### 8.13.3.2 unsigned char qftop::set_control::action

Definition at line 41 of file qftop_application.hpp.

#### 8.13.3.3 unsigned char qftop::set_control::byte

Definition at line 43 of file qftop_application.hpp.

#### 8.13.3.4 unsigned char qftop::set_control::fill

Definition at line 38 of file qftop_application.hpp.

#### 8.13.3.5 unsigned char qftop::set_control::has_credentials

Definition at line 40 of file qftop_application.hpp.

#### 8.13.3.6 unsigned char qftop::set_control::has_iid

Definition at line 39 of file qftop_application.hpp.

#### 8.13.3.7 unsigned char qftop::set_control::mode

Definition at line 37 of file qftop_application.hpp.

## 8.14 qftop::tty Class Reference

```
#include <qftop_tty.hpp>
```

### Public Member Functions

- tty (const std::string &device_name, std::function< void(std::unique_ptr< qfTopMessage >)> on_new_message_callback)
- void send_message (std::unique_ptr< qfTopMessage > message)

  *Send a QFTOP message.*
- void start_reading ()

  *Start reading data from tty.*
- void stop_reading ()

  *Stop reading data from tty.*

### 8.14.1 Detailed Description

Definition at line 27 of file qftop_tty.hpp.

## 8.14.2 Constructor & Destructor Documentation

### 8.14.2.1 qftop::tty::tty ( const std::string & *device_name,* std::function< void(std::unique_ptr< **qfTopMessage** >)> *on_new_message_callback* )

Constructor

**Parameters**

| | | |
|---|---|---|
| in | *device_name* | name of tty device |
| in | *on_new_-message_-callback* | callback called when a QFTOP message has been received |

Definition at line 10 of file qftop_tty.cpp.

### 8.14.3 Member Function Documentation

#### 8.14.3.1 void qftop::tty::send_message ( std::unique_ptr< qfTopMessage > *message* )

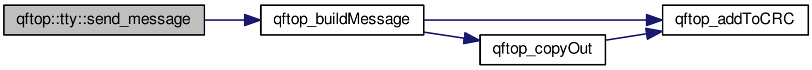
Send a QFTOP message.

**Parameters**

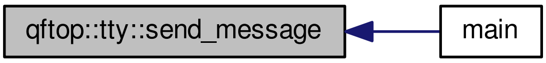
| | | |
|---|---|---|
| in | *message* | QFTOP message to be sent |

Definition at line 26 of file qftop_tty.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

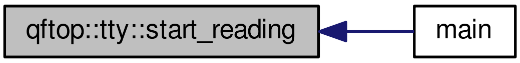


#### 8.14.3.2 void qftop::tty::start_reading ( )

Start reading data from tty.

Definition at line 49 of file qftop_tty.cpp.

Here is the caller graph for this function:

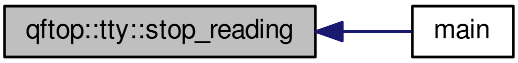


#### 8.14.3.3 void qftop::tty::stop_reading ( )

Stop reading data from tty.

Definition at line 59 of file qftop_tty.cpp.

Here is the caller graph for this function:



## 8.15 qftop::write_response_callback Class Reference

Write Response Callback Interface.

```
#include <qftop_application.hpp>
```

### Public Member Functions

- virtual void on_success ()=0
- virtual void on_error ()=0

### 8.15.1 Detailed Description

Write Response Callback Interface.

Definition at line 167 of file qftop_application.hpp.

### 8.15.2 Member Function Documentation

#### 8.15.2.1 virtual void qftop::write_response_callback::on_error ( ) [pure virtual]

#### 8.15.2.2 virtual void qftop::write_response_callback::on_success ( ) [pure virtual]

## 8.16 qftop::write_without_cred Struct Reference

Write Without Credentials Request.

`#include <qftop_application.hpp>`

### Public Member Functions

- std::vector< unsigned char > to_bytes ()

### Public Attributes

- unsigned char length
- unsigned char fragment_header
- struct set_control control
- unsigned char element_id
- unsigned char attribute_count
- struct attribute_list attribute_list

### Friends

- std::ostream & operator<< (std::ostream &, const write_without_cred &)

### 8.16.1 Detailed Description

Write Without Credentials Request.

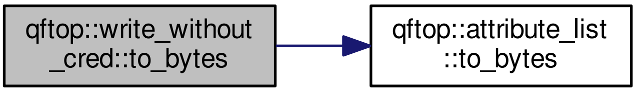
Definition at line 72 of file qftop_application.hpp.

### 8.16.2 Member Function Documentation

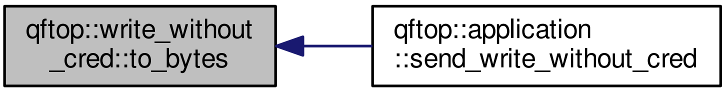
#### 8.16.2.1 std::vector<unsigned char> qftop::write_without_cred::to_bytes ( )

Definition at line 79 of file qftop_application.hpp.

Here is the call graph for this function:


qftop::write_without _cred::to_bytes
qftop::attribute_list ::to_bytes

Here is the caller graph for this function:



### 8.16.3 Friends And Related Function Documentation

#### 8.16.3.1 std::ostream& operator<< ( std::ostream & , const write_without_cred & ) [friend]

Definition at line 41 of file qftop_application.cpp.

### 8.16.4 Member Data Documentation

#### 8.16.4.1 unsigned char qftop::write_without_cred::attribute_count

Definition at line 77 of file qftop_application.hpp.

#### 8.16.4.2 struct attribute_list qftop::write_without_cred::attribute_list

Definition at line 78 of file qftop_application.hpp.

#### 8.16.4.3 struct set_control qftop::write_without_cred::control

Definition at line 75 of file qftop_application.hpp.

#### 8.16.4.4 unsigned char qftop::write_without_cred::element_id

Definition at line 76 of file qftop_application.hpp.

#### 8.16.4.5 unsigned char qftop::write_without_cred::fragment_header

Definition at line 74 of file qftop_application.hpp.

#### 8.16.4.6 unsigned char qftop::write_without_cred::length

Definition at line 73 of file qftop_application.hpp.

# Chapter 9

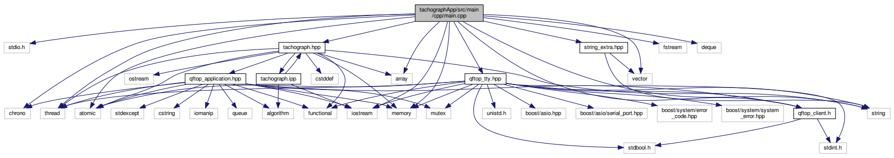
# File Documentation

## 9.1 dox/mainpage.dox File Reference

## 9.2 tachographApp/src/main/cpp/main.cpp File Reference

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <thread>
#include <memory>
#include <atomic>
#include <string>
#include <chrono>
#include <vector>
#include <deque>
#include <array>
#include "qftop_tty.hpp"
#include "qftop_client.h"
#include "tachograph.hpp"
#include "string_extra.hpp"
```

Include dependency graph for main.cpp:



### Functions

- int main ()

## 9.3 main.cpp

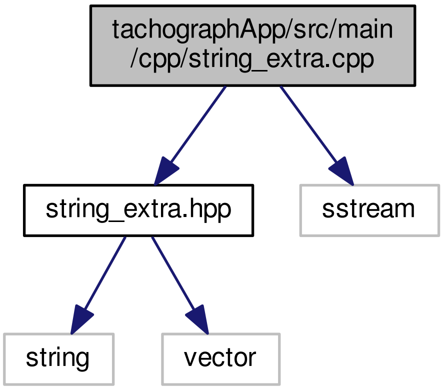
```

#include <stdio.h>
#include <iostream>
#include <fstream>
```

```
#include <thread>
#include <memory>
#include <atomic>
#include <string>
#include <chrono>
#include <vector>
#include <deque>
#include <array>
#include "qftop_tty.hpp"
#include "qftop_client.h"
#include "tachograph.hpp"
#include "string_extra.hpp"

int main() {
    using string_extra::has_prefix;
    using string_extra::split_on_whitespace;
    using string_extra::hex_to_bytes;

    std::cout << "Tachograph" << std::endl;
    std::shared_ptr<std::ofstream> output_stream_ptr = std::make_shared<std::ofstream>();
    output_stream_ptr->open("log.txt");

    std::unique_ptr<qftop::tty> tty_ptr;
    const auto qftop_application_ptr = std::make_shared<qftop::application>(
        [&](auto message_ptr) { tty_ptr->send_message(std::move(message_ptr)); },
output_stream_ptr);

    std::string user_input;
    const std::string default_tty_device = "/dev/ttyUSB0";

    while (true) {
        std::cout << "Input tty device (or <> to use " << default_tty_device << ")" << std::endl;
        std::getline(std::cin, user_input);
        std::string tty_device;
        if (user_input.empty()) {
            tty_device = default_tty_device;
        } else {
            tty_device = user_input;
        }
        try {
            tty_ptr = std::make_unique<qftop::tty>(tty_device, [&](std::unique_ptr<qfTopMessage> msg_ptr) {
                qftop_application_ptr->push_message(std::move(msg_ptr));
            });
        } catch (const std::exception &e) {
            std::cout << "Failed to initialize tty" << std::endl;
            continue;
        }
        std::cout << "Initialized tty" << std::endl;
        break;
    }

    std::cout << "Input ’i’ to write data with counter and increment counter" << std::endl;
    std::cout << "Input ’r’ to read data " << std::endl;
    std::cout << "Input ’w <tachograph_payload> [<dsrc_security_data>]’ to write RTM data" << std::endl;
    std::cout << "Input ’q’ to finish" << std::endl;

    int counter = 0;

    const std::size_t dsrc_security_data_length = 16;
    const std::size_t tachograph_payload_length = 67;
    const auto tachograph_application_ptr =
        std::make_unique<tachograph::application<tachograph_payload_length, dsrc_security_data_length>>(
            qftop_application_ptr, output_stream_ptr);
    qftop_application_ptr->start_polling();
    tty_ptr->start_reading();

    while (true) {
        std::getline(std::cin, user_input);
        if (user_input == "q") {
            break;
        } else if (user_input == "r") {
            tachograph_application_ptr->read_rtm_data(
                [](std::array<unsigned char, tachograph_payload_length> tachograph_payload,
                   std::array<unsigned char, dsrc_security_data_length> dsrc_security_data) {
                    std::cout << "Tachograph Payload: ";
                    for (const auto &character : tachograph_payload) {
                        std::cout << std::setfill(’0’) << std::setw(2) << std::hex << (int)character;
                    }
                    std::cout << std::endl << "Dsrc Security Data: ";
                    for (const auto &character : dsrc_security_data) {
                        std::cout << std::setfill(’0’) << std::setw(2) << std::hex << (int)character;
                    }
                    std::cout << std::endl;
                });
        } else if (has_prefix(user_input, "w")) {
            auto tokens = split_on_whitespace(user_input);

            std::array<unsigned char, dsrc_security_data_length> dsrc_security_data;
            int tokens_count = tokens.size();
            if (tokens_count == 2) {
                dsrc_security_data.fill(0);
            } else if (tokens_count == 3) {
                auto dsrc_security_data_string = tokens[2];
                if (dsrc_security_data_string.size() == (dsrc_security_data_length * 2)) {
                    auto bytes = hex_to_bytes(dsrc_security_data_string);
                    for (int i = 0; i < (int)dsrc_security_data_length; i += 1) {
                        dsrc_security_data[i] = bytes[i];
                    }
                } else {
                    std::cout << "<dsrc_security_data> length incorrect. Should be 32 hex chars. Was "
                              << dsrc_security_data_string.size() << std::endl;
                    continue;
                }
            } else {
                std::cout << "Incorrect input" << std::endl;
                continue;
            }

            std::array<unsigned char, tachograph_payload_length> tachograph_payload;
            auto tachograph_payload_string = tokens[1];
            if (tachograph_payload_string.size() == (tachograph_payload_length * 2)) {
                auto bytes = hex_to_bytes(tachograph_payload_string);
                for (int i = 0; i < (int)tachograph_payload_length; i += 1) {
                    tachograph_payload[i] = bytes[i];
                }
            } else {
                std::cout << "<tachograph_payload> length incorrect. Should be 134 hex chars. Was "
                          << tachograph_payload_string.size() << std::endl;
                continue;
            }

            tachograph_application_ptr->write_rtm_data(tachograph_payload, dsrc_security_data);
        } else if (user_input == "i") {
            counter += 1;
            std::array<unsigned char, tachograph_payload_length> tachograph_payload;
            for (int i = 0; i < (int)tachograph_payload_length; i += 4) {
                tachograph_payload[i] = 0x7a;
                tachograph_payload[i + 1] = 0xc0;
                tachograph_payload[i + 2] = 0xbe;
                tachograph_payload[i + 3] = counter;
            }

            tachograph_payload[64] = 0xff;
            tachograph_payload[65] = 0xff;
            tachograph_payload[66] = 0xff;

            std::array<unsigned char, dsrc_security_data_length> dsrc_security_data;

            for (int i = 0; i < (int)dsrc_security_data_length; i += 4) {
                dsrc_security_data[i] = 0x01;
                dsrc_security_data[i + 1] = 0x23;
                dsrc_security_data[i + 2] = 0x45;
                dsrc_security_data[i + 3] = 0x67;
            }

            tachograph_application_ptr->write_rtm_data(tachograph_payload, dsrc_security_data);
        }
    }

    tty_ptr->stop_reading();
    qftop_application_ptr->stop_polling();
    output_stream_ptr->close();

    return EXIT_SUCCESS;
}
```

## 9.4 tachographApp/src/main/cpp/string_extra.cpp File Reference

```
#include "string_extra.hpp"
#include <sstream>
```

Include dependency graph for string_extra.cpp:



## Namespaces

- string_extra

## Functions

- bool string_extra::has_prefix (std::string string_to_check, std::string prefix)

  *Check if string has prefix.*
- std::vector< std::string > string_extra::split_on_whitespace (std::string string_to_split)

  *Split string on whitespace.*
- std::vector< unsigned char > string_extra::hex_to_bytes (std::string hex_string)

  *Convert hex string to bytes.*

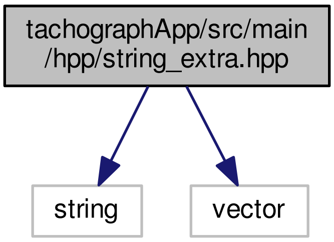
## 9.5 string_extra.cpp

```
#include "string_extra.hpp"
#include <sstream>

namespace string_extra {

bool has_prefix(std::string string_to_check, std::string prefix) {
    return std::equal(prefix.begin(), prefix.end(), string_to_check.begin());
}

std::vector<std::string> split_on_whitespace(std::string string_to_split) {
    std::string buffer;
    std::stringstream stream(string_to_split);
    std::vector<std::string> tokens;
    while (stream >> buffer) {
        tokens.push_back(buffer);
    }
    return tokens;
}

std::vector<unsigned char> hex_to_bytes(std::string hex_string) {
    std::stringstream stream(hex_string);
    std::vector<unsigned char> bytes;
    unsigned int buffer;
    unsigned int offset = 0;
    while (offset < hex_string.length()) {
        stream.str(std::string());
        stream.clear();
        stream << std::hex << hex_string.substr(offset, 2);
        stream >> std::hex >> buffer;
        bytes.push_back(static_cast<unsigned char>(buffer));
        offset += 2;
    }
    return bytes;
}
}
```

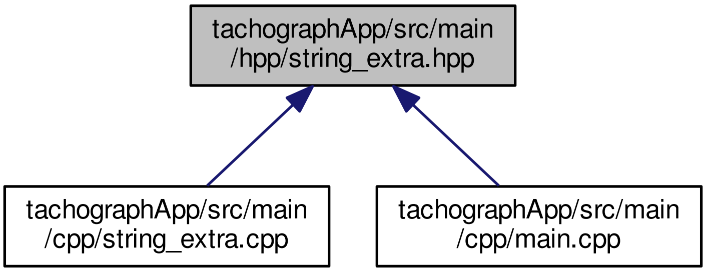
## 9.6 tachographApp/src/main/hpp/string_extra.hpp File Reference

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

Include dependency graph for string_extra.hpp:



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



### Namespaces

- string_extra

### Functions

- bool string_extra::has_prefix (std::string string_to_check, std::string prefix)

  *Check if string has prefix.*
- std::vector< std::string > string_extra::split_on_whitespace (std::string string_to_split)

  *Split string on whitespace.*
- std::vector< unsigned char > string_extra::hex_to_bytes (std::string hex_string)

  *Convert hex string to bytes.*

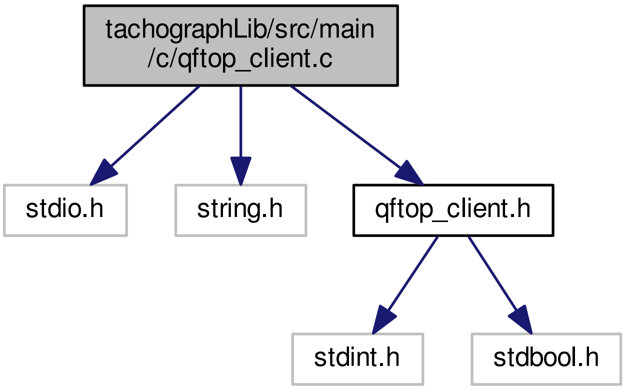
## 9.7 string_extra.hpp

```

#ifndef STRING_EXTRA_HPP
#define STRING_EXTRA_HPP

#include <string>
#include <vector>

namespace string_extra {
bool has_prefix(std::string string_to_check, std::string prefix);

std::vector<std::string> split_on_whitespace(std::string string_to_split);

std::vector<unsigned char> hex_to_bytes(std::string hex_string);
} // namespace string_extra
#endif // STRING_EXTRA_HPP

```

## 9.8 tachographLib/src/main/c/qftop_client.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "qftop_client.h"
```

Include dependency graph for qftop_client.c:



### Functions

- uint8_t ∗ qftop_copyIn (struct qfTopMessage ∗msg, uint8_t ∗in, uint8_t ∗out, int len) *helper to copy a string of bytes into another vector based on the state of a messages*

- uint8_t ∗ qftop_copyOut (struct qfTopMessage ∗msg, uint8_t ∗out, uint8_t ∗in, int len)

  *helper to copy a string of bytes from another vector based on the state of a messages*
- void qftop_addToCRC (struct qfTopMessage ∗msg, uint8_t ch)

  *Modify crc calculation for a new byte.*
- void qftop_clear (struct qfTopMessage ∗msg)

  *Convenience function to zero / reset a message.*
- unsigned int qftop_buildMessage (struct qfTopMessage ∗msg)

  *Function to build a byte stream ready for transmission based on a message.*
- void qftop_ack (struct qfTopMessage ∗msg)

  *Convenience function to build an ACK message.*
- void qftop_application (struct qfTopMessage ∗msg)

  *Convenience function to initialise an application message (dsrc req)*
- void qftop_addParameter (struct qfTopMessage ∗msg, uint8_t p)

  *Function for adding a single parameter to a QFTop message.*
- uint16_t qftop_extractMessage (struct qfTopMessage ∗msg_out, struct qfTopMessage ∗msg_in)

  *Function to build an internal message based on bytes in another message.*
- int qftop_parse (struct qfTopMessage ∗msg, uint8_t cr)

  *Function to parse a new byte into a message being received.*

## 9.8.1 Function Documentation

### 9.8.1.1 uint8_t ∗ qftop_copyIn ( struct **qfTopMessage** ∗ *msg,* uint8_t ∗ *in,* uint8_t ∗ *out,* int *len* )

helper to copy a string of bytes into another vector based on the state of a messages

**Parameters**

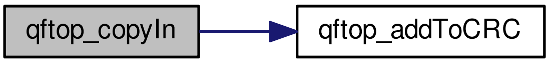
| | | |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |
| in | *in* | input bytes |
| in | *out* | output bytes |
| in | *len* | no of bytes to process |

**Returns**

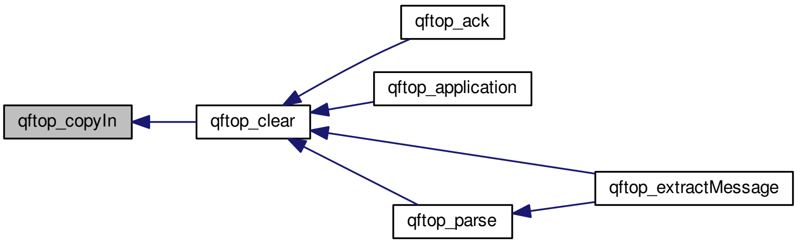
len final pointer

Definition at line 23 of file qftop_client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.8.1.2 uint8_t ∗ qftop_copyOut ( struct qfTopMessage ∗ *msg,* uint8_t ∗ *out,* uint8_t ∗ *in,* int *len* )

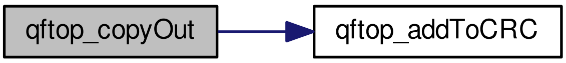
helper to copy a string of bytes from another vector based on the state of a messages

**Parameters**

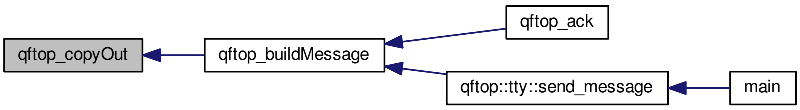
| | | |
|---|---|---|
| in | *msg* | pointer to qfTopMessage |
| in | *in* | input bytes |
| in | *out* | output bytes |
| in | *len* | No of bytes to copy |

Definition at line 53 of file qftop_client.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 9.9 qftop_client.c

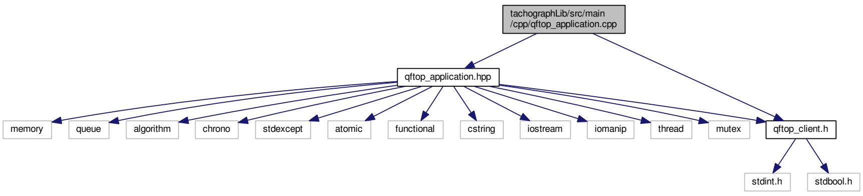
```
00001 #include <stdio.h>
00002 #include <string.h>
```

```c
#include "qftop_client.h"

uint8_t *qftop_copyIn(struct qfTopMessage *msg, uint8_t *in, uint8_t *out, int len);
uint8_t *qftop_copyOut(struct qfTopMessage *msg, uint8_t *out, uint8_t *in, int len);

void qftop_addToCRC(struct qfTopMessage *msg, uint8_t ch) {
    ch = (ch ^ (uint8_t)(msg->msg.CRC.word & 0xFF));
    ch = (ch ^ (uint8_t)(ch << 4));
    msg->msg.CRC.word = (msg->msg.CRC.word >> 8) ^ ((uint16_t)ch << 8) ^ ((uint16_t)ch << 3) ^ ((uint16_t)ch >> 4);
}

uint8_t *qftop_copyIn(struct qfTopMessage *msg, uint8_t *in, uint8_t *out, int len) {
    uint8_t *in_p = in;
    uint8_t *out_p = out;
    uint8_t v;
    for (int i = 0; i < len;) {
        v = *in_p++;
        if (v == qftop_preamble) {
            if (msg->pre_escape) {
                msg->pre_escape = false;
                continue;
            }
            msg->pre_escape = true;
        } else {
            msg->pre_escape = false;
        }
        *out_p++ = v;
        qftop_addToCRC(msg, v);
        i += 1;
    }
    return in_p;
}

uint8_t *qftop_copyOut(struct qfTopMessage *msg, uint8_t *out, uint8_t *in, int len) {
    uint8_t *in_p = in;
    uint8_t *out_p = out;
    uint8_t v;
    for (int i = 0; i < len; i += 1) {
        v = *in_p++;
        *out_p++ = v;
        qftop_addToCRC(msg, v);
        if (v == qftop_preamble) {
            *out_p++ = qftop_preamble;
        }
    }
    return out_p;
}

void qftop_clear(struct qfTopMessage *msg) {
    uint8_t zero[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    qftop_copyIn(msg, zero, msg->msg.HEADER.bytes, (int)3);
    qftop_copyIn(msg, zero, msg->msg.PDU.PARAMETERS, (int)10);
    msg->msg.PDU.messageType = 0;
    msg->msg.CRC.word = crc_init;
    msg->msg.CRC_REC.word = 0xFFFF;
    msg->message_length = 0;
    msg->pre_escape = false;
    msg->head_read = false;
    msg->head_start = false;
}

unsigned int qftop_buildMessage(struct qfTopMessage *msg) {
    uint8_t *m = msg->message;
    uint8_t v;
    msg->msg.CRC.word = crc_init;
    v = msg->msg.HEADER.bytes[0];
    *m++ = v;
    if (v == qftop_preamble) {
        *m++ = qftop_preamble;
    }
    v = msg->msg.HEADER.bytes[1];
    *m++ = v;
    qftop_addToCRC(msg, v);
    if (v == qftop_preamble) {
        *m++ = qftop_preamble;
    }
    v = msg->msg.HEADER.bytes[2];
    *m++ = v;
    qftop_addToCRC(msg, v);
    if (v == qftop_preamble) {
        *m++ = qftop_preamble;
    }
    m = qftop_copyOut(msg, m, msg->msg.PDU.bytes, msg->msg.HEADER.length);
    *m++ = msg->msg.CRC_REC.bytes[0] = msg->msg.CRC.bytes[0];
    *m++ = msg->msg.CRC_REC.bytes[1] = msg->msg.CRC.bytes[1];
    msg->message_length = m - msg->message;
    return msg->message_length;
}

void qftop_ack(struct qfTopMessage *msg) {
    qftop_clear(msg);
    msg->pre_escape = false, msg->head_read = false, msg->head_start = false;
    msg->message_length = 0;
    msg->msg.HEADER.preamble = qftop_preamble;
    msg->msg.HEADER.frameType = 1; // ACK;
    msg->msg.PDU.messageType = 0;
    msg->msg.HEADER.length = (uint8_t)0;
    msg->msg.CRC.word = crc_init;
    qftop_buildMessage(msg);
}

void qftop_application(struct qfTopMessage *msg) {
    qftop_clear(msg);
    msg->pre_escape = false;
    msg->head_read = false;
    msg->message_length = 0;
    msg->msg.HEADER.preamble = qftop_preamble;
    msg->msg.HEADER.frameType = Application;
    msg->msg.PDU.messageType = dsrc_l7_req;
    msg->msg.HEADER.length = (uint8_t)1;
    msg->msg.CRC.word = crc_init;
}

void qftop_addParameter(struct qfTopMessage *msg, uint8_t p) {
    msg->msg.PDU.PARAMETERS[msg->msg.HEADER.length++ - 1] = 0xFF & p;
    qftop_addToCRC(msg, p);
}

uint16_t qftop_extractMessage(struct qfTopMessage *msg_out, struct qfTopMessage *msg_in) {
    uint16_t rc = 0;
    qftop_clear(msg_out);

    for (int i = 0; i < msg_in->message_length; i += 1) {
        rc = qftop_parse(msg_out, msg_in->message[i]);
        msg_out->message[i] = msg_in->message[i];
    }
    msg_out->message_length = msg_in->message_length;
    return rc;
}

int qftop_parse(struct qfTopMessage *msg, uint8_t cr) {
    if (cr == qftop_preamble) {
        if (msg->pre_escape) {
            msg->pre_escape = false;
        } else {
            msg->pre_escape = true;
            return 0;
        }
    } else if (msg->pre_escape) {
        qftop_clear(msg);
        msg->message[msg->message_length++] = qftop_preamble;
        msg->message[msg->message_length++] = cr;
        msg->head_start = true;
        msg->head_read = false;
        msg->pre_escape = false;
        return 0;
    }

    if (!msg->head_start) {
        return 0;
    }

    msg->message[msg->message_length++] = cr;
    if (msg->message_length > maximumQFTOPFrameSize) {
        msg->head_start = false;
        msg->message_length = 0;
        return -1;
    }

    if (!msg->head_read) {
        msg->msg.HEADER.bytes[0] = msg->message[0];
        msg->msg.HEADER.bytes[1] = msg->message[1];
        msg->msg.HEADER.bytes[2] = msg->message[2];
        qftop_addToCRC(msg, msg->msg.HEADER.bytes[1]);
        qftop_addToCRC(msg, msg->msg.HEADER.bytes[2]);
        msg->head_read = true;
        return 0;
    } else if (msg->message_length == msg->msg.HEADER.length + 5) {
        for (int i = 0; i < msg->msg.HEADER.length; i += 1) {
            msg->msg.PDU.bytes[i] = msg->message[i + 3];
            qftop_addToCRC(msg, msg->msg.PDU.bytes[i]);
        }
        msg->msg.CRC_REC.bytes[1] = msg->message[msg->message_length - 1];
        msg->msg.CRC_REC.bytes[0] = msg->message[msg->message_length - 2];
        return 1;
    }
    return 0;
}
```

## 9.10 tachographLib/src/main/cpp/qftop_application.cpp File Reference

```
#include "qftop_application.hpp"
#include "qftop_client.h"
```

Include dependency graph for qftop_application.cpp:



### Namespaces

- qftop

### Functions

- void qftop::print_message (std::ostream &out, const qfTopMessage ∗rhs)

  *Print QFTOP message to stream.*
- std::ostream & qftop::operator<< (std::ostream &out, const set_control &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const attribute_list &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const write_without_cred &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const get_control &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const read_without_cred &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const read_without_cred_response &rhs)

## 9.11 qftop_application.cpp

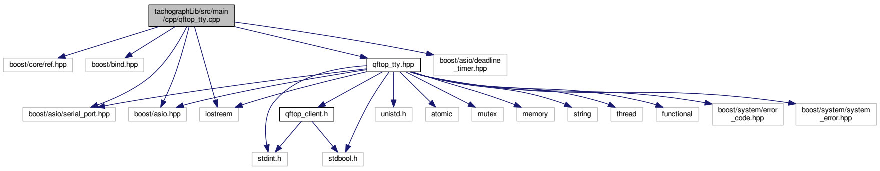
```
#include "qftop_application.hpp"
#include "qftop_client.h"

namespace qftop {

void print_message(std::ostream &out, const qfTopMessage *rhs) {
    out << "Preamble: " << std::setw(2) << std::hex << (int)rhs->msg.HEADER.preamble << std::endl
        << "Sequence number: " << std::setw(2) << std::hex << (int)rhs->msg.HEADER.sequence << std::endl
        << "Frame type: " << (int)rhs->msg.HEADER.frameType << std::endl
        << "Status: " << (int)rhs->msg.HEADER.status << std::endl
        << "SYN: " << (int)rhs->msg.HEADER.syn << std::endl
        << "Length: " << (int)rhs->msg.HEADER.length << std::endl
        << "Message type: " << (int)rhs->msg.PDU.messageType << std::endl
        << "Parameters: ";
    for (int i = 0; i < (int)(rhs->msg.HEADER.length - 1); i += 1) {
        out << std::setfill('0') << std::setw(2) << std::hex << (int)rhs->msg.PDU.PARAMETERS[i];
    }
    out << std::endl;
    out << "CRC: " << std::setfill('0') << std::setw(2) << std::hex << rhs->msg.CRC.word << std::endl;
}

std::ostream &operator<<(std::ostream &out, const set_control &rhs) {
    return out << "Mode: " << (int)rhs.mode << std::endl
               << "Fill: " << (int)rhs.fill << std::endl
               << "Has IID: " << (int)rhs.has_iid << std::endl
               << "Has Credentials: " << (int)rhs.has_credentials << std::endl
               << "Action: " << (int)rhs.action << std::endl;
}

std::ostream &operator<<(std::ostream &out, const attribute_list &rhs) {
    out << "Attribute ID: " << (int)rhs.attribute_id << std::endl
        << "Container type: " << std::setfill('0') << std::setw(2) << std::hex << (int)rhs.container_type << std::endl
        << "Attribute Value: ";
    for (auto &&character : rhs.attribute_value) {
        out << std::setfill('0') << std::setw(2) << std::hex << (int)character;
    }
    out << std::endl;
    return out;
}

std::ostream &operator<<(std::ostream &out, const write_without_cred &rhs) {
    return out << "Length: " << (int)rhs.length << std::endl
               << "Fragment header: " << std::setfill('0') << std::setw(2) << std::hex << (int)rhs.fragment_header
               << std::endl
               << rhs.control << "Element ID: " << (int)rhs.element_id << std::endl
               << "Attribute count: " << (int)rhs.attribute_count << std::endl
               << rhs.attribute_list;
}

std::ostream &operator<<(std::ostream &out, const get_control &rhs) {
    return out << "Mode: " << (int)rhs.mode << std::endl
               << "Has attribute list: " << (int)rhs.has_attribute_list << std::endl
               << "Has IID: " << (int)rhs.has_iid << std::endl
               << "Has Credentials: " << (int)rhs.has_credentials << std::endl
               << "Action: " << (int)rhs.action << std::endl;
}

std::ostream &operator<<(std::ostream &out, const read_without_cred &rhs) {
    return out << "Length: " << (int)rhs.length << std::endl
               << "Fragment header: " << std::setfill('0') << std::setw(2) << std::hex << (int)rhs.fragment_header
               << std::endl
               << rhs.control << "Element ID: " << (int)rhs.element_id << std::endl
               << "Attribute count: " << (int)rhs.attribute_count << std::endl
               << "Attribute list: " << (int)rhs.attribute_list[0] << std::endl;
}

std::ostream &operator<<(std::ostream &out, const read_without_cred_response &rhs) {
    out << "Length: " << (int)rhs.header.length << std::endl
        << "Fragment header: " << std::setfill('0') << std::setw(2) << std::hex << (int)rhs.header.fragment_header
        << std::endl
        << rhs.header.control << "Element ID: " << (int)rhs.header.element_id << std::endl
        << "Attribute count: " << (int)rhs.header.attribute_count << std::endl
        << "Attribute ID: " << (int)rhs.header.attribute_id << std::endl
        << "Container ID: " << (int)rhs.header.container_id << std::endl
        << "Attribute value: " << std::endl;
    for (auto character : rhs.attribute_value) {
        out << std::setfill('0') << std::setw(2) << std::hex << (int)character;
    }
    out << std::endl;
    return out;
}

application::application(std::function<void(std::unique_ptr<qfTopMessage>)> on_message_write_callback,
                         std::shared_ptr<std::ostream> output_stream)
    : on_message_write_callback(on_message_write_callback), output_stream(std::move(output_stream)) {
    auto x = this->create_empty_queue();
    this->queue = std::move(x);
```

```cpp
    this->run = std::make_shared<std::atomic<bool>>(false);
    this->sequence_counter.store((unsigned char)0);
}

void application::push_message(std::unique_ptr<qfTopMessage> message) {
    this->queue_lock.lock();
    this->queue->push(std::move(message));
    this->queue_lock.unlock();
}

bool application::has_messages() {
    this->queue_lock.lock();
    bool queue_not_empty = !this->queue->empty();
    this->queue_lock.unlock();
    return queue_not_empty;
}

std::unique_ptr<qfTopMessage> application::pop_message() {
    this->queue_lock.lock();
    auto message = std::move(this->queue->front());
    this->queue->pop();
    this->queue_lock.unlock();
    return std::move(message);
}

void application::send_message(std::unique_ptr<qfTopMessage> message_ptr) {
    print_message(*this->output_stream, &(*message_ptr));
    this->on_message_write_callback(std::move(message_ptr));
}

void application::clear_messages() {
    this->queue_lock.lock();
    auto empty_queue = this->create_empty_queue();
    std::swap(this->queue, empty_queue);
    this->queue_lock.unlock();
}

std::unique_ptr<qfTopMessage> application::create_message() {
    auto message_ptr = std::make_unique<qfTopMessage>();
    auto message_raw_ptr = message_ptr.get();
    qftop_application(message_raw_ptr);
    return std::move(message_ptr);
}

void application::start_polling() {
    this->input_thread_lock.lock();
    bool was_running = this->run->exchange(true);
    if (!was_running) {
        this->input_thread =
            std::make_unique<std::thread>(std::bind(&application::process_input, this, this->run, this->output_stream));
    }
    this->input_thread_lock.unlock();
}

void application::stop_polling() {
    this->input_thread_lock.lock();
    bool was_running = this->run->exchange(false);
    if (was_running) {
        this->input_thread->join();
    }
    this->input_thread_lock.unlock();
}

void application::send_write_without_cred(unsigned char element_id, unsigned char attribute_id,
                                          std::vector<unsigned char> attribute_value,
                                          std::shared_ptr<write_response_callback> on_write_response_callback) {
    unsigned char tapdu_set_request = 4;

    struct write_without_cred request;
    request.length = sizeof(request);
    request.fragment_header = 0x91;
    request.control.action = tapdu_set_request;
    request.control.has_credentials = 0;
    request.control.has_iid = 0;
    request.control.fill = 0;
    request.control.mode = 1;
    request.element_id = element_id;
    request.attribute_count = 1;
    request.attribute_list.attribute_id = attribute_id;
    request.attribute_list.container_type = 10;
    request.attribute_list.attribute_value = std::move(attribute_value);

    *(this->output_stream) << "----Sending SET.request----" << std::endl << request << "----------------" << std::endl;

    auto request_bytes = request.to_bytes();

    const unsigned char sequence_number = this->sequence_counter.fetch_add(1, std::memory_order_seq_cst);
    this->callbacks_lock.lock();
    this->write_response_callbacks.push(std::move(on_write_response_callback));
    this->callbacks_lock.unlock();

    const unsigned char message_type_in = QFTOP_DSRC_L7_REQ;
    this->send_message(request_bytes, message_type_in, sequence_number);
    delay();
}

void application::send_read_without_cred(unsigned char element_id, unsigned char attribute_id,
                                         std::shared_ptr<read_response_callback> on_read_response_callback) {
    unsigned char a[] = {attribute_id};
    unsigned char tapdu_get_request = 6;
    struct read_without_cred get_request;
    get_request.length = 5;
    get_request.fragment_header = 0x91;
    get_request.control.action = tapdu_get_request;
    get_request.control.has_credentials = 0;
    get_request.control.has_iid = 0;
    get_request.control.has_attribute_list = 1;
    get_request.control.mode = 0;
    get_request.element_id = element_id;
    get_request.attribute_count = 1;
    get_request.attribute_list[0] = a[0];

    *(this->output_stream) << "----Sending GET.request----" << std::endl
                           << get_request << "----------------" << std::endl;

    const unsigned char sequence_number = this->sequence_counter.fetch_add(1, std::memory_order_seq_cst);

    this->callbacks_lock.lock();
    this->read_response_callbacks.push(std::move(on_read_response_callback));
    this->callbacks_lock.unlock();

    const unsigned char command_type_in = QFTOP_DSRC_L7_REQ;
    this->send_message(get_request.to_bytes(), command_type_in, sequence_number);

    delay();
}

void application::send_message(std::vector<unsigned char> raw_message, uint8_t message_type_in,
                               unsigned char sequence_number) {
    auto message_request_ptr = this->create_message();

    message_request_ptr->msg.PDU.messageType = message_type_in;
    message_request_ptr->msg.HEADER.syn = 1;
    message_request_ptr->msg.HEADER.sequence = sequence_number;

    auto message_request_raw_ptr = message_request_ptr.get();
    for (auto &&character : raw_message) {
        qftop_addParameter(message_request_raw_ptr, character);
    }
    this->send_message(std::move(message_request_ptr));
};

std::unique_ptr<std::queue<std::unique_ptr<qfTopMessage>>> application::create_empty_queue() {
    return std::move(std::make_unique<std::queue<std::unique_ptr<qfTopMessage>>>());
}

void application::delay() {
    const int sleep_time_ms = 100;
    std::this_thread::sleep_for(std::chrono::milliseconds(sleep_time_ms));
}

void application::process_input(std::shared_ptr<std::atomic<bool>> run, std::shared_ptr<std::ostream> output_stream) {
    while (run->load()) {
        this->delay();
        while (this->has_messages()) {
            *(output_stream) << "Has message(s)" << std::endl;
            auto message_ptr = this->pop_message();

            if (message_ptr->msg.PDU.messageType == QFTOP_DSRC_L7_RESP) {
                *(output_stream) << "Message is L7 RESP" << std::endl;
                unsigned char sequence_number = message_ptr->msg.HEADER.sequence;
                *(output_stream) << "seq num = " << (int)sequence_number << std::endl;

                print_message(*(output_stream), &(*message_ptr));
                // TODO: sanity check parameters length
                const unsigned char action = (message_ptr->msg.PDU.PARAMETERS[2] >> 4);
                const unsigned char tapdu_get_response = 7;
                const unsigned char tapdu_set_response = 5;
                if (action == tapdu_get_response) {
                    this->callbacks_lock.lock();
                    if (!this->read_response_callbacks.empty()) {
                        const auto callback = this->read_response_callbacks.front();
                        *(output_stream) << "Found read callback" << std::endl;

                        read_without_cred_response response;

                        std::memcpy(&response.header, message_ptr->msg.PDU.PARAMETERS, sizeof(response.header));

                        const int header_length = sizeof(response.header.bytes);
                        const int attribute_value_start = header_length;
                        const int attribute_value_end = attribute_value_start + response.header.length;

                        std::vector<unsigned char> attribute_value;
                        for (int i = attribute_value_start; i != attribute_value_end; i += 1) {
                            attribute_value.push_back(message_ptr->msg.PDU.PARAMETERS[i]);
                        }

                        *(output_stream) << response << std::endl;

                        // TODO: Error checking
                        callback->on_success(std::move(attribute_value));

                        this->read_response_callbacks.pop();
                    } else {
                        *(output_stream) << "Could not find read callback" << std::endl;
                    }
                    this->callbacks_lock.unlock();
                } else if (action == tapdu_set_response) {
                    this->callbacks_lock.lock();
                    if (!this->write_response_callbacks.empty()) {
                        const auto callback = this->write_response_callbacks.front();
                        *(output_stream) << "Found write callback" << std::endl;

                        // write_without_cred_response response;
                        // TODO: Error checking
                        callback->on_success();
                        this->write_response_callbacks.pop();
                    } else {
                        *(output_stream) << "Could not find write callback" << std::endl;
                    }
                    this->callbacks_lock.unlock();
                } else {
                    *(output_stream) << "Unknown action: " << std::setfill('0') << std::setw(2) << std::hex
                                     << (int)action << std::endl;
                }
            } else {
                *(output_stream) << "Message was ignored" << std::endl;
            }
        }
    }
}
} // namespace qftop
```

## 9.12 tachographLib/src/main/cpp/qftop_tty.cpp File Reference

```
#include <boost/core/ref.hpp>
#include <boost/bind.hpp>
#include <boost/asio/serial_port.hpp>
#include <boost/asio.hpp>
#include <boost/asio/deadline_timer.hpp>
#include "qftop_tty.hpp"
#include <iostream>
```

Include dependency graph for qftop_tty.cpp:



## Namespaces

- qftop

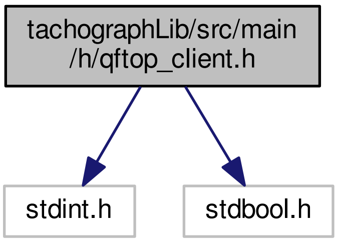
## 9.13 qftop_tty.cpp

```
#include <boost/core/ref.hpp>
#include <boost/bind.hpp>
#include <boost/asio/serial_port.hpp>
#include <boost/asio.hpp>
#include <boost/asio/deadline_timer.hpp>
#include "qftop_tty.hpp"
#include <iostream>

namespace qftop {
tty::tty(const std::string &device_name, std::function<void(std::unique_ptr<qfTopMessage>)> on_new_message_callback)
    : io_service()
    , port(io_service, device_name)
    , on_new_message_callback(on_new_message_callback)
    , run(std::make_shared<std::atomic<bool>>(false)) {
    {
        using boost::asio::serial_port_base;
        boost::system::error_code error_code;
        this->port.set_option(serial_port_base::baud_rate(115200));
        this->port.set_option(serial_port_base::parity(serial_port_base::parity::type::none));
        this->port.set_option(serial_port_base::character_size(8));
        this->port.set_option(serial_port_base::stop_bits(serial_port_base::stop_bits::type::one));
        this->port.set_option(serial_port_base::flow_control(serial_port_base::flow_control::type::none));
    }
}

void tty::send_message(std::unique_ptr<qfTopMessage> message_ptr) {
    qftop_buildMessage(&(*message_ptr));

    using boost::system::error_code;
    using boost::asio::buffer;

    const char wake_up_sequence[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    error_code wake_up_sequence_error_code;
    const int wakeup_bytes_written =
        this->port.write_some(buffer(wake_up_sequence, sizeof(wake_up_sequence)), wake_up_sequence_error_code);
    if (wake_up_sequence_error_code || (wakeup_bytes_written != sizeof(wake_up_sequence))) {
        std::cerr << "Error writing wake up bytes" << std::endl;
        return;
    }

    error_code message_error_code;
    const int message_bytes_written =
        this->port.write_some(buffer(message_ptr->message, message_ptr->message_length), message_error_code);
    if (message_error_code || (message_bytes_written != message_ptr->message_length)) {
        std::cerr << "Error writing message" << std::endl;
    }
}

void tty::start_reading() {
    this->input_thread_lock.lock();
    bool was_running = this->run->exchange(true);
    if (was_running) {
        return;
    }
    this->input_thread = std::make_unique<std::thread>(std::bind(&tty::read_messages, this));
    this->input_thread_lock.unlock();
}

void tty::stop_reading() {
    this->input_thread_lock.lock();
    bool was_running = this->run->exchange(false);
    if (was_running) {
        this->input_thread->join();
    }
    this->input_thread_lock.unlock();
}

void tty::read_messages() {
    std::vector<char> input_buffer(1024);
    std::atomic<bool> data_available(false);
    const unsigned int timeout = 100;
    boost::asio::deadline_timer timer(this->io_service);
    std::unique_ptr<qfTopMessage> current_message_ptr = std::make_unique<qfTopMessage>();

    while (this->run->load()) {
        this->port.async_read_some(boost::asio::buffer(input_buffer),
                                   boost::bind(&tty::read_callback, this, boost::ref(data_available), boost::ref(timer),
                                               boost::asio::placeholders::error,
                                               boost::asio::placeholders::bytes_transferred));
        timer.expires_from_now(boost::posix_time::milliseconds(timeout));
        timer.async_wait(
            boost::bind(&tty::timeout_callback, this, boost::ref(this->port), boost::asio::placeholders::error));
        this->io_service.run(); // Blocks until all async callbacks are finished
        this->io_service.reset();
        if (!(data_available.load())) {
            continue;
        }

        for (auto &&character : input_buffer) {
            uint8_t return_code = qftop_parse(&(*current_message_ptr), character);
            if (return_code > 0) {
                this->on_new_message_callback(std::move(current_message_ptr));
                current_message_ptr = std::make_unique<qfTopMessage>();
            }
        }
    }
}

void tty::read_callback(std::atomic<bool> &data_available, boost::asio::deadline_timer &timeout,
                        const boost::system::error_code &error_code, std::size_t bytes_transferred) {
    if (error_code || !bytes_transferred) {
        data_available.store(false);
        return;
    }
    timeout.cancel(); // Will cause wait_callback to fire with an error
    data_available.store(true);
}

void tty::timeout_callback(boost::asio::serial_port &serial_port, const boost::system::error_code & error_code) {
    if (error_code) {
        // Data was read and this timeout was canceled
        return;
    }
    serial_port.cancel(); // Will cause read_callback to fire with an error
}
} // namespace qftop
```

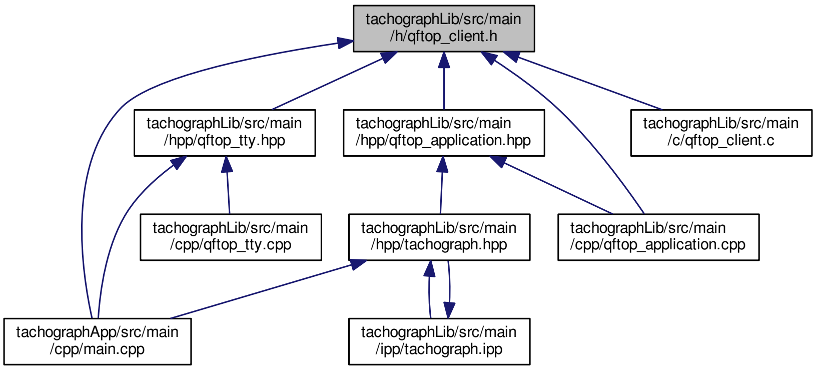
## 9.14 tachographLib/src/main/h/qftop_client.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for qftop_client.h:



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



## Classes

- struct qfTopMSG

  *Structure of QFTop Message.*
- union qfTopMSG::header
- union qfTopMSG::pdu
- union qfTopMSG::crc
- struct qfTopMessage

  *State holder of QFTop Message plus state info when parsing.*

## Macros

- #define MAXIMUMQFTOPFRAMESIZE 200

## Enumerations

- enum qftop_cmd_type_t {
  QFTOP_ECHO_REQ = 0x00, QFTOP_ECHO_RESP = 0x80, QFTOP_ACK = 0x01, QFTOP_NACK = 0x02,
  QFTOP_MMI_REQ = 0x30, QFTOP_INIT_NOTIFICATION = 0x31, QFTOP_TRANSP_RESP = 0x33, QFT-

OP_TRANSP_REQ = 0x34,
QFTOP_REGISTER_APP_REQ = 0x36, QFTOP_REGISTER_APP = 0x37, QFTOP_TEST_REQ = 0x38, QFTOP_TEST_RESP = 0x39,
QFTOP_PERS_REQ = 0x3A, QFTOP_PERS_RESP = 0x3B, QFTOP_DSRC_L7_REQ = 0x3C, QFTOP_DSRC_L7_RESP = 0x3D,
QFTOP_TRACE_LOG_REQ = 0xF0, QFTOP_TRACE_LOG_RESP = 0xF1 }

*Types of QFTOP messages.*

- enum qftop_Types {
Application = 0, ACK = 1, NACK = 2, dsrc_l7_req = 0x3C,
dsrc_l7_resp = 0x3D, crc_init = 0x6363, qftop_preamble = 0xB5, maximumQFTOPFrameSize = MAXIMUMQFTOPFRAMESIZE }

  *Types of QFTOP messages.*

### Functions

- int qftop_parse (struct qfTopMessage ∗msg, uint8_t cr)

  *Function to parse a new byte into a message being received.*
- uint16_t qftop_extractMessage (struct qfTopMessage ∗msg_out, struct qfTopMessage ∗msg_in)

  *Function to build an internal message based on bytes in another message.*
- void qftop_addToCRC (struct qfTopMessage ∗msg, uint8_t ch)

  *Modify crc calculation for a new byte.*
- void qftop_addParameter (struct qfTopMessage ∗msg, uint8_t p)

  *Function for adding a single parameter to a QFTop message.*
- void qftop_clear (struct qfTopMessage ∗msg)

  *Convenience function to zero / reset a message.*
- void qftop_application (struct qfTopMessage ∗msg)

  *Convenience function to initialise an application message (dsrc req)*
- void qftop_ack (struct qfTopMessage ∗msg)

  *Convenience function to build an ACK message.*
- unsigned int qftop_buildMessage (struct qfTopMessage ∗msg)

  *Function to build a byte stream ready for transmission based on a message.*

## 9.15 qftop_client.h

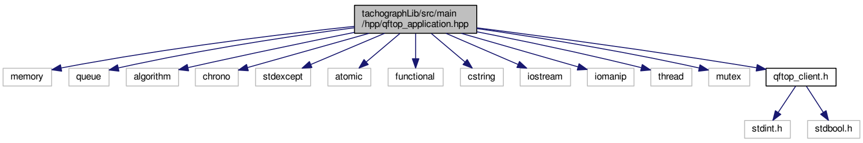
```

#ifndef QFTOP_CLIENT_H
#define QFTOP_CLIENT_H

#ifdef __cplusplus
extern "C" {
#endif

#define MAXIMUMQFTOPFRAMESIZE 200

#include <stdint.h>
#include <stdbool.h>

typedef enum {
    QFTOP_ECHO_REQ = 0x00,
    QFTOP_ECHO_RESP = 0x80,
    QFTOP_ACK = 0x01,
    QFTOP_NACK = 0x02,
    QFTOP_MMI_REQ = 0x30,
    QFTOP_INIT_NOTIFICATION = 0x31,
    QFTOP_TRANSP_RESP = 0x33,
    QFTOP_TRANSP_REQ = 0x34,
    QFTOP_REGISTER_APP_REQ = 0x36,
    QFTOP_REGISTER_APP = 0x37,
    QFTOP_TEST_REQ = 0x38,
    QFTOP_TEST_RESP = 0x39,
    QFTOP_PERS_REQ = 0x3A,
    QFTOP_PERS_RESP = 0x3B,
    QFTOP_DSRC_L7_REQ = 0x3C,
    QFTOP_DSRC_L7_RESP = 0x3D,
    QFTOP_TRACE_LOG_REQ = 0xF0,
    QFTOP_TRACE_LOG_RESP = 0xF1
} qftop_cmd_type_t;

enum qftop_Types {
    Application = 0,
    ACK = 1,
    NACK = 2,
    dsrc_l7_req = 0x3C,
    dsrc_l7_resp = 0x3D,
    crc_init = 0x6363,
    qftop_preamble = 0xB5,
    maximumQFTOPFrameSize = MAXIMUMQFTOPFRAMESIZE
};
struct qfTopMSG {
    union header {
        struct {
            unsigned char preamble : 8;
            unsigned char sequence : 4;
            unsigned char frameType : 2;
            unsigned char status : 1;
            unsigned char syn : 1;
            unsigned char length : 8;
        };
        uint8_t bytes[3];
    } HEADER;
    union pdu {
        struct {
            uint8_t messageType;
            uint8_t PARAMETERS[maximumQFTOPFrameSize];
        };
        uint8_t bytes[maximumQFTOPFrameSize + 1];
    } PDU;
    union crc {
        uint8_t bytes[2];
        uint16_t word;
    } CRC,
        CRC_REC;
};
struct qfTopMessage {
    struct qfTopMSG msg;
    bool pre_escape;
    bool head_start;
    bool head_read;
    uint8_t message[maximumQFTOPFrameSize + 10];
    unsigned int message_length;
};

int qftop_parse(struct qfTopMessage *msg, uint8_t cr);

uint16_t qftop_extractMessage(struct qfTopMessage *msg_out, struct
qfTopMessage *msg_in);

void qftop_addToCRC(struct qfTopMessage *msg, uint8_t ch);

void qftop_addParameter(struct qfTopMessage *msg, uint8_t p);

void qftop_clear(struct qfTopMessage *msg);

void qftop_application(struct qfTopMessage *msg);

void qftop_ack(struct qfTopMessage *msg);

unsigned int qftop_buildMessage(struct qfTopMessage *msg);

#ifdef __cplusplus
}
#endif

#endif    // QFTOP_CLIENT_H
/* @} */
```

## 9.16 tachographLib/src/main/hpp/qftop_application.hpp File Reference

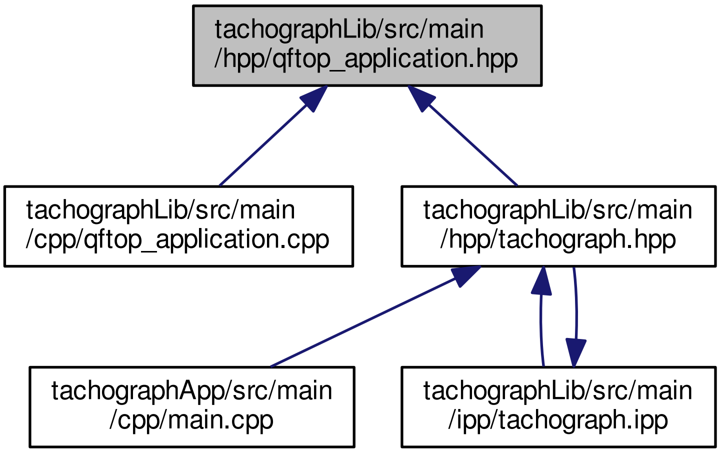
```
#include <memory>
```

```
#include <queue>
#include <algorithm>
#include <chrono>
#include <stdexcept>
#include <atomic>
#include <functional>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <thread>
#include <mutex>
#include "qftop_client.h"
```

Include dependency graph for qftop_application.hpp:



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



## Classes

- struct qftop::set_control

  *Set Control Data.*
- struct qftop::attribute_list

  *Attribute List.*
- struct qftop::write_without_cred

  *Write Without Credentials Request.*
- struct qftop::get_control

*Get Control Data.*

- struct qftop::read_without_cred

    *Read Without Credentials Request.*

- struct qftop::read_without_cred_response

    *Read Without Credentials Response.*

- class qftop::write_response_callback

    *Write Response Callback Interface.*

- class qftop::read_response_callback

    *Read Response Callback Interface.*

- class qftop::application

    *QFTOP Client.*

### Namespaces

- qftop

### Functions

- void qftop::print_message (std::ostream &out, const qfTopMessage *rhs)

    *Print QFTOP message to stream.*

- std::ostream & qftop::operator<< (std::ostream &out, const set_control &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const attribute_list &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const write_without_cred &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const get_control &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const read_without_cred &rhs)
- std::ostream & qftop::operator<< (std::ostream &out, const read_without_cred_response &rhs)

## 9.17 qftop_application.hpp

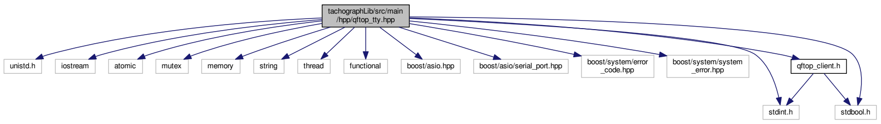
```
#ifndef QFTOP_APPLICATION_HPP
#define QFTOP_APPLICATION_HPP

#include <memory>
#include <queue>
#include <algorithm>
#include <chrono>
#include <stdexcept>
#include <atomic>
#include <functional>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <thread>
#include <mutex>
#include "qftop_client.h"

namespace qftop {
void print_message(std::ostream &out, const qfTopMessage *rhs);

struct set_control {
    union {
        struct {
            unsigned char mode : 1;
            unsigned char fill : 1;
            unsigned char has_iid : 1;
            unsigned char has_credentials : 1;
            unsigned char action : 4;
        };
        unsigned char byte;
    };
    friend std::ostream &operator<<(std::ostream &, const set_control &);
};

std::ostream &operator<<(std::ostream &out, const set_control &rhs);

```

```
struct attribute_list {
    unsigned char attribute_id;
    unsigned char container_type;
    std::vector<unsigned char> attribute_value;
    std::vector<unsigned char> to_bytes() {
        std::vector<unsigned char> bytes;
        bytes.push_back(attribute_id);
        bytes.push_back(container_type);
        bytes.insert(bytes.end(), attribute_value.begin(), attribute_value.end());
        return std::move(bytes);
    }
    friend std::ostream &operator<<(std::ostream &, const attribute_list &);
};

std::ostream &operator<<(std::ostream &out, const attribute_list &rhs);

struct write_without_cred {
    unsigned char length;
    unsigned char fragment_header;
    struct set_control control;
    unsigned char element_id;
    unsigned char attribute_count;
    struct attribute_list attribute_list;
    std::vector<unsigned char> to_bytes() {
        std::vector<unsigned char> bytes;
        bytes.push_back(length);
        bytes.push_back(fragment_header);
        bytes.push_back(control.byte);
        bytes.push_back(element_id);
        bytes.push_back(attribute_count);
        auto attribute_list_bytes = attribute_list.to_bytes();
        bytes.insert(bytes.end(), attribute_list_bytes.begin(), attribute_list_bytes.end());
        return std::move(bytes);
    }
    friend std::ostream &operator<<(std::ostream &, const write_without_cred &);
};

std::ostream &operator<<(std::ostream &out, const write_without_cred &rhs);

struct get_control {
    union {
        struct {
            unsigned char mode : 1;
            unsigned char has_attribute_list : 1;
            unsigned char has_iid : 1;
            unsigned char has_credentials : 1;
            unsigned char action : 4;
        };
        unsigned char byte;
    };
    friend std::ostream &operator<<(std::ostream &, const get_control &);
};

std::ostream &operator<<(std::ostream &out, const get_control &rhs);

struct read_without_cred {
    union {
        struct {
            unsigned char length;
            unsigned char fragment_header;
            struct get_control control;
            unsigned char element_id;
            unsigned char attribute_count;
            unsigned char attribute_list[1];
        };
        unsigned char bytes[6];
    };
    std::vector<unsigned char> to_bytes() {
        std::vector<unsigned char> bytes;
        for (auto &&character : this->bytes) {
            bytes.push_back(character);
        }
        return std::move(bytes);
    }

    friend std::ostream &operator<<(std::ostream &, const read_without_cred &);
};

std::ostream &operator<<(std::ostream &out, const read_without_cred &rhs);

struct read_without_cred_response {
    union {
        struct {
            unsigned char length;
            unsigned char fragment_header;
            struct get_control control;
            unsigned char element_id;
            unsigned char attribute_count;
            unsigned char attribute_id;
            unsigned char container_id;
        };
        unsigned char bytes[sizeof(unsigned char) * 6 + sizeof(get_control)];
    } header;
    std::vector<unsigned char> attribute_value;
    friend std::ostream &operator<<(std::ostream &, const
read_without_cred_response &);
};

std::ostream &operator<<(std::ostream &out, const
read_without_cred_response &rhs);

class write_response_callback {
public:
    virtual void on_success() = 0;
    virtual void on_error() = 0;
};

class read_response_callback {
public:
    virtual void on_success(std::vector<unsigned char> attribute_value) = 0;
    virtual void on_error() = 0;
};

class application {
public:
    application(std::function<void(std::unique_ptr<qfTopMessage>)> on_message_write_callback,
                std::shared_ptr<std::ostream> output_stream);

    void push_message(std::unique_ptr<qfTopMessage> message);

    void start_polling();

    void stop_polling();

    void send_write_without_cred(unsigned char element_id, unsigned char
attribute_id,
                                 std::vector<unsigned char> attribute_value,
                                 std::shared_ptr<write_response_callback> on_write_response_callback);
    void send_read_without_cred(unsigned char element_id, unsigned char attribute_id,
                                std::shared_ptr<read_response_callback> on_read_response_callback);

private:
    std::queue<std::shared_ptr<write_response_callback>> write_response_callbacks;
    std::queue<std::shared_ptr<read_response_callback>> read_response_callbacks;
    std::function<void(std::unique_ptr<qfTopMessage>)> on_message_write_callback;
    std::shared_ptr<std::ostream> output_stream;
    std::shared_ptr<std::atomic<bool>> run;
    std::unique_ptr<std::thread> input_thread;
    std::mutex queue_lock;
    std::mutex callbacks_lock;
    std::unique_ptr<std::queue<std::unique_ptr<qfTopMessage>>> queue;
    std::mutex input_thread_lock;
    std::atomic<unsigned char> sequence_counter;

    static std::unique_ptr<std::queue<std::unique_ptr<qfTopMessage>>> create_empty_queue();

    static void delay();

    void process_input(std::shared_ptr<std::atomic<bool>> run, std::shared_ptr<std::ostream> output_stream)
;

    void send_message(std::vector<unsigned char> raw_message, uint8_t message_type_in, unsigned char
sequence_number);

    bool has_messages();

    std::unique_ptr<qfTopMessage> pop_message();

    void send_message(std::unique_ptr<qfTopMessage> message_ptr);

    void clear_messages();

    static std::unique_ptr<qfTopMessage> create_message();
};
} // namespace qftop
#endif // QFTOP_APPLICATION_HPP
```

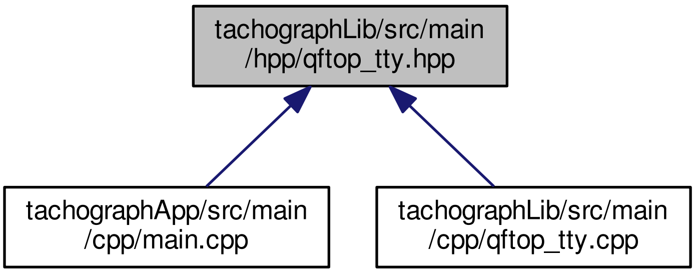
## 9.18 tachographLib/src/main/hpp/qftop_tty.hpp File Reference

```
#include <unistd.h>
#include <iostream>
#include <atomic>
#include <mutex>
#include <memory>
#include <string>
#include <thread>
#include <functional>
#include <boost/asio.hpp>
#include <boost/asio/serial_port.hpp>
#include <boost/system/error_code.hpp>
#include <boost/system/system_error.hpp>
#include <stdint.h>
#include <stdbool.h>
#include "qftop_client.h"
```

Include dependency graph for qftop_tty.hpp:



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



### Classes

- class qftop::tty

### Namespaces

- qftop

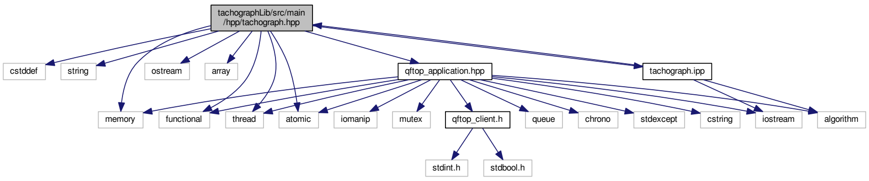
## 9.19 qftop_tty.hpp

```
#ifndef QFTOP_TTY_HPP
#define QFTOP_TTY_HPP

#include <unistd.h>
#include <iostream>
#include <atomic>
#include <mutex>
#include <memory>
#include <string>
#include <thread>
#include <functional>
#include <boost/asio.hpp>
#include <boost/asio/serial_port.hpp>
#include <boost/system/error_code.hpp>
#include <boost/system/system_error.hpp>
#include <stdint.h>
#include <stdbool.h>

#include "qftop_client.h"

namespace qftop {

class tty {
public:
    tty(const std::string &device_name, std::function<void(std::unique_ptr<qfTopMessage>)>
on_new_message_callback);

    void send_message(std::unique_ptr<qfTopMessage> message);

    void start_reading();

    void stop_reading();

private:
    std::function<void(std::unique_ptr<qfTopMessage>)> on_new_message_callback;
    boost::asio::io_service io_service;
    boost::asio::serial_port port;
    std::unique_ptr<std::thread> input_thread;
    std::shared_ptr<std::atomic<bool>> run;
    std::mutex input_thread_lock;
    void read_messages();
    void read_callback(std::atomic<bool> &data_available, boost::asio::deadline_timer &timeout,
                       const boost::system::error_code &error, std::size_t bytes_transferred);
    void timeout_callback(boost::asio::serial_port &serial_port, const boost::system::error_code &
error_code);
};
} // namespace qftop

#endif // QFTOP_TTY_HPP
```

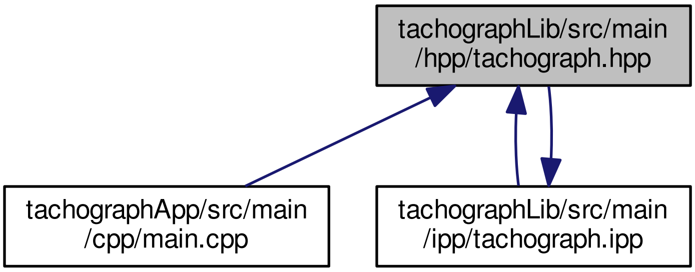
## 9.20 tachographLib/src/main/hpp/tachograph.hpp File Reference

```
#include <cstddef>
#include <string>
#include <memory>
#include <ostream>
#include <array>
#include <functional>
#include <thread>
#include <atomic>
#include "qftop_application.hpp"
#include "tachograph.ipp"
```

Include dependency graph for tachograph.hpp:



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



## Classes

- class tachograph::application< TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH >

  *Tachograph Client.*

## Namespaces

- tachograph

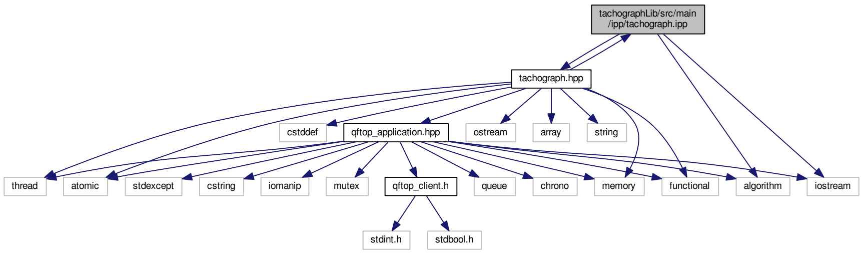
## 9.21 tachograph.hpp

```
#ifndef TACHOGRAPH_H
#define TACHOGRAPH_H

#include <cstddef>
#include <string>
#include <memory>
#include <ostream>
#include <array>
#include <functional>
#include <thread>
#include <atomic>
#include "qftop_application.hpp"

namespace tachograph {
template <std::size_t TACHOGRAPH_PAYLOAD_LENGTH, std::size_t DSRC_SECURITY_DATA_LENGTH>
class application {
public:
    application(std::shared_ptr<qftop::application> qftop_application_ptr,
                std::shared_ptr<std::ostream> output_stream_ptr);

    void read_rtm_data(
        std::function<void(std::array<unsigned char, TACHOGRAPH_PAYLOAD_LENGTH> tachograph_payload,
                           std::array<unsigned char, DSRC_SECURITY_DATA_LENGTH> dsrc_security_data)> callback);

    void write_rtm_data(std::array<unsigned char, TACHOGRAPH_PAYLOAD_LENGTH> tachograph_payload,
                        std::array<unsigned char, DSRC_SECURITY_DATA_LENGTH> dsrc_security_data);

private:
    std::shared_ptr<qftop::application> application_ptr;
    std::unique_ptr<std::thread> input_thread_ptr;
    std::shared_ptr<std::ostream> output_stream_ptr;
    std::shared_ptr<std::atomic<bool>> run_ptr;

    static void process_input(std::shared_ptr<std::atomic<bool>> run_ptr,
                              std::shared_ptr<qftop::application> application_ptr,
                              std::shared_ptr<std::ostream> output_stream_ptr);
};
}
#include "tachograph.ipp"

#endif // TACHOGRAPH_H
```

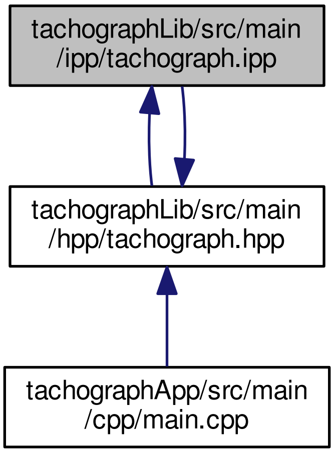
## 9.22 tachographLib/src/main/ipp/tachograph.ipp File Reference

```
#include "tachograph.hpp"
#include <algorithm>
#include <iostream>
```

Include dependency graph for tachograph.ipp:



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



### Namespaces

- tachograph

## 9.23 tachograph.ipp

```
#include "tachograph.hpp"
#include <algorithm>
#include <iostream>

namespace tachograph {

template <std::size_t TACHOGRAPH_PAYLOAD_LENGTH, std::size_t DSRC_SECURITY_DATA_LENGTH>
application<TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH>::application(
    std::shared_ptr<qftop::application> qftop_application_ptr, std::shared_ptr<std::ostream> output_stream_ptr)
    : application_ptr(qftop_application_ptr), output_stream_ptr(output_stream_ptr) {
}

template <std::size_t TACHOGRAPH_PAYLOAD_LENGTH, std::size_t DSRC_SECURITY_DATA_LENGTH>
void application<TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH>::read_rtm_data(
    std::function<void(std::array<unsigned char, TACHOGRAPH_PAYLOAD_LENGTH> tachograph_payload,
                       std::array<unsigned char, DSRC_SECURITY_DATA_LENGTH> dsrc_security_data)> callback) {
    uint8_t element_id_rtm = 1;
    uint8_t attribute_id_rtm_data = 1;

    class internal_callback : public qftop::read_response_callback {
    public:
        internal_callback(
            std::function<void(std::array<unsigned char, TACHOGRAPH_PAYLOAD_LENGTH> tachograph_payload,
                               std::array<unsigned char, DSRC_SECURITY_DATA_LENGTH> dsrc_security_data)> callback)
            : callback(std::move(callback)) {
        }
        void on_success(std::vector<unsigned char> attribute_value) override {
            std::cout << "Successful read" << std::endl;
            std::array<unsigned char, (int)TACHOGRAPH_PAYLOAD_LENGTH> tachograph_payload;
            std::copy_n(attribute_value.begin(), (int)TACHOGRAPH_PAYLOAD_LENGTH, tachograph_payload.begin());
            std::array<unsigned char, (int)DSRC_SECURITY_DATA_LENGTH> dsrc_security_data;
            std::copy_n(attribute_value.begin() + 1 + (int)TACHOGRAPH_PAYLOAD_LENGTH, (int)DSRC_SECURITY_DATA_LENGTH,
                        dsrc_security_data.begin());
            this->callback(std::move(tachograph_payload), std::move(dsrc_security_data));
        }
        void on_error() override {
            std::cout << "Erronous read" << std::endl;
        }

    private:
        std::function<void(std::array<unsigned char, TACHOGRAPH_PAYLOAD_LENGTH> tachograph_payload,
                           std::array<unsigned char, DSRC_SECURITY_DATA_LENGTH> dsrc_security_data)> callback;
    };

    auto internal_callback_ptr = std::make_shared<internal_callback>(std::move(callback));

    this->application_ptr->send_read_without_cred(element_id_rtm, attribute_id_rtm_data,
                                                  std::move(internal_callback_ptr));
}

template <std::size_t TACHOGRAPH_PAYLOAD_LENGTH, std::size_t DSRC_SECURITY_DATA_LENGTH>
void application<TACHOGRAPH_PAYLOAD_LENGTH, DSRC_SECURITY_DATA_LENGTH>::write_rtm_data(
    std::array<unsigned char, TACHOGRAPH_PAYLOAD_LENGTH> tachograph_payload,
    std::array<unsigned char, DSRC_SECURITY_DATA_LENGTH> dsrc_security_data) {
    const std::size_t DSRC_SECURITY_DATA_SIZE_LENGTH = 1;
    const std::size_t RTM_DATA_LENGTH =
        TACHOGRAPH_PAYLOAD_LENGTH + DSRC_SECURITY_DATA_SIZE_LENGTH + DSRC_SECURITY_DATA_LENGTH;
    uint8_t element_id_rtm = 1;
    uint8_t attribute_id_rtm_data = 1;

    class internal_callback : public qftop::write_response_callback {
    public:
        void on_success() override {
            std::cout << "Successful write" << std::endl;
        }
        void on_error() override {
            std::cout << "Erronous write" << std::endl;
        }
    };

    std::vector<unsigned char> rtm_data;
    for (auto &&character : tachograph_payload) {
        rtm_data.push_back(character);
    }
    rtm_data.push_back((unsigned char)DSRC_SECURITY_DATA_SIZE_LENGTH);
    for (auto &&character : dsrc_security_data) {
        rtm_data.push_back(character);
    }

    auto internal_callback_ptr = std::make_shared<internal_callback>();

    this->application_ptr->send_write_without_cred(element_id_rtm, attribute_id_rtm_data, std::move(rtm_data),
                                                   std::move(internal_callback_ptr));
}
}
```

# Index

# References

Commission, E., ‘Regulation (eu) no 165/2014 of the european parliament and of the council’, 2004.

Commission, E., ‘Commission implementing regulation (eu) 2016/799 of 18 march 2016 implementing regulation (eu) no 165/2014 of the european parliament and of the council laying down the requirements for the construction, testing, installation, operation and repair of tachographs and their components’. 2011.

## List of abbreviations and definitions

**ADEV** Allan Deviation

**AKOS** Agency for Communication Networks and Services of the Republic of Slovenia

**C-ITS** Cooperative Intelligent Transport Systems

**CNIT** Consorzio Nazionale Interuniversitario per le Telecomunicazioni

**COTS** Commercial Off-The-Shelf

**CS** Commercial Service

**DSRC** Dedicated Short Range Communications

**EKF** Extended Kalmann Filtering

**GNSS** Global Navigation Satellite System

**GSA** European GNSS Agency

**IMU** Inertial Mounted Unit

**ITS** Intelligent Transportation System

**IV** Intelligent Vehicle

**NMA** Navigation Message Authentication

**OBU** On Board Unit

**OS** Open Service

**PF** Particle Filtering

**PoC** Proof of Concept

**PRS** Public Regulated Service

**PVT** Position Velocity and Time

**RF** Radio Frequency

**ST** Smart Tachograph

**SDR** Software Defined Radio

**TESLA** Timed Efficient Stream Loss-tolerant Authentication

**VU** Vehicle Unit

## List of figures

***Europe Direct is a service to help you find answers to your questions about the European Union.***

**Freephone number** (*)**:**

**00 800 6 7 8 9 10 11**

(*) The information given is free, as are most calls (though some operators, phone boxes or hotels may charge you).

More information on the European Union is available on the internet (http://europa.eu).

**HOW TO OBTAIN EU PUBLICATIONS**

**Free publications:**

- one copy:
  via EU Bookshop (http://bookshop.europa.eu);
- more than one copy or posters/maps:
  from the European Union’s representations (http://ec.europa.eu/represent_en.htm);
  from the delegations in non-EU countries (http://eeas.europa.eu/delegations/index_en.htm);
  by contacting the Europe Direct service (http://europa.eu/europedirect/index_en.htm) or
  calling 00 800 6 7 8 9 10 11 (freephone number from anywhere in the EU) (*).

  (*) The information given is free, as are most calls (though some operators, phone boxes or hotels may charge you).

**Priced publications:**

- via EU Bookshop (http://bookshop.europa.eu).

LB-NA-28378-EN-N

## JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

**EU Science Hub**
ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

Joint Research Centre

EU Science Hub

Publications Office

doi:10.2788/866989

ISBN 978-92-79-64805-2