VehicleComm - Usage instructions

Contents

[Introduction 1](#__RefHeading__334_975607888)

[The Makefile 1](#__RefHeading__336_975607888)

[The header files 1](#__RefHeading__338_975607888)

[The c++ file 2](#__RefHeading__340_975607888)

[Initializing the interface 2](#__RefHeading__342_975607888)

[Using the interface 2](#__RefHeading__344_975607888)

[Receiving responses 3](#__RefHeading__346_975607888)

[De-initializing the interface 4](#__RefHeading__348_975607888)

# Introduction

The following steps are needed to use the VehicleComm-module.

# The Makefile

In Makefile add the package:

packages := vehicle\_comm\_client

# The header files

Include the following .hpp-files in your code:

#include <vehicle\_comm\_client.hpp>

#include <vehicle\_comm\_messages.hpp>

And add a reference to a VehicleCommClient-object as a member:

class YourObject {

private:

vc::VehicleCommClient& vc\_;

...

}

The next step is to add members in the class for callbacks to receive responses and events:

void EventCallback(vc::MessageBase \*message);

void ResponseCallback(vc::MessageBase \*m, vc::ReturnValue status);

# The c++ file

The user of VehicleComm should get a reference to the singleton object of VehicleCommClient by using the GetInstance-member and store it in the parent class:

YourObject::YourObject()

: vc\_(vc::VehicleCommClient::GetInstance()) {

}

# Initializing the interface

To user starts by initializing the VehicleCommClient:

if (vc\_.Init() != RET\_OK) {

LOG(LOG\_ERR, Failed to init VehicleComm-client!\n");

return TSRET\_ERR\_EXTERNAL;

}

Then register the callback-functions defined in the header-file, using lambda-functions:

vc\_.RegisterResponseCallback([this](vc::MessageBase \*message, ReturnValue status){return TSServer::ResponseCallback(message, status);});

vc\_.RegisterEventCallback([this](vc::MessageBase \*message){return YourObject::EventCallback(message);});

Or by using STL:

vc\_.RegisterResponseCallback(std::bind(&YourObject::ResponseCallback, this, std::placeholders::\_1));

vc\_.RegisterEventCallback(std::bind(&YourObject::EventCallback, this, std::placeholders::\_1));

# Using the interface

Now we are ready to interact with the VehicleCommClient object.

Call request-functions:

Example 1:

vc::ReqHornNLight req;

req.mode = vc::HL\_REQUEST\_HORN\_AND\_LIGHT;

vc\_.Request\_HornAndLight(&req, session\_id);

Example 2:

vc\_.Request\_SwitchToFlashBootloader(session\_id);

# Receiving responses

In the response-callback that was registered, we can handle each response corresponding to a request that was sent.

Example 1 (Print all data in the payload):

void ResponseCallback(MessageBase \*m, ReturnValue status)

{

if (m == NULL) || (status != vc::RET\_OK))

return;

PrintMessage(m);

}

Example 2 (HornNLight):

void ResponseCallback(vc::MessageBase \*m, vc::ReturnValue status)

{

if (m == NULL)

return;

if ((m->type\_ == vc::MESSAGE\_RESPONSE) &&

((VCResponseID)m->id\_ == vc::RES\_HORNNLIGHT)) {

if (status == vc::RET\_OK) {

ResHornNLight& res =

((vc::Message<vc::ResHornNLight>\*)m)->message;

LOG(LOG\_DEBUG, "mode = %d\n", res.return\_code);

} else {

LOG(LOG\_ERR, "Error received! Response is invalid.”);

}

}

}

# De-initializing the interface

Nothing is needed.