D5.7

TeamMate Extension SDK

Project Number:	690705
Classification **Deliverable No.:**	Public D5.7
Work Package(s): **Document Version:** **Issue Date:** **Document Timescale:** Start of the Document: Final version due:	WP5 Vs. 0.1 31.08.2019 Project Start Date: September 1, 2016 Month 35 Month 36
Deliverable Overview:	**Main document:** D5.7
Compiled by:	Elham Fathiazar, HMT
Authors:	Elisa Landini, REL Andrea Castellano, REL Stefan Suck, OFF Daniel Twumasi, HMT Elham Fathiazar, HMT
Technical Approval:	Fabio Tango, CRF
Issue Authorisation:	Andreas Lüdtke, OFF

DISTRIBUTION LIST		
Copy type[1]	Company and Location	Recipient
T	AutoMate Consortium	all AutoMate Partners

[1] Copy types: E=Email, C=Controlled copy (paper), D=electronic copy on Disk or other medium, T=Team site (Sharepoint)

RECORD OF REVISION		
Date	Status Description	Author
23.07.2019	Deliverable structure	Elham Fathiazar
16.08.2019	Content for TeamMate Component Extension SDK	Elham Fathiazar Stefan Suck Daniel Twumasi
27.08.2019	Content for Mobile App Extension SDK	Elisa Ladini Andrea Castellano
28.08.2019	Content for Component Validation Framework and Results for Evaluation	Elham Fathiazar Elisa Ladini
30.08.2019	Review	Fabio Tango

Table of Contents

List of Abbreviations

AR	*Augmented Reality*
CAN	*Controller Area Network*
DIR	*Driver Intention Recognition*
DPU	*Data Processing Unit*
FW	*Firmware*
HW	*Hardware*
HMI	*Human Machine Interaktion*
IDE	*Integrated Development Environment*
I/O	*Input/Output*
protobuf	*Google Protocol Buffers*
SDK	*Software Development Kit*
SUS	*Driver Intention Recognition*
TAM	*Technology Acceptance Model*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

List of Figures

List of Tables

Executive Summary

In deliverable D5.7, we present the Extension SDKs, which enables third parties to extend the TeamMate system. Two complementary extensions, namely “TeamMate Component Extension SDK" and "Mobile app Extension SDK", are designed to enable third parties to have access to the enablers of the TeamMate System as well as the output interface and data acquired by the TeamMate system. The “TeamMate Component Extension SDK" provides tools to replace or upgrade some enablers of the TeamMate system and thereby modifying the intrinsic functionalities of TeamMate system, for example general decision making or strategy planning. "Mobile app Extension SDK" provides access to the CAN bus of a vehicle to easily create mobile apps that use this data. Therefore, this SDK enables third parties to develop new mobile applications that exploit the potential of the functionality of the TeamMate system without having a deep knowledge of the hardware and firmware of the vehicle.

The deliverable also describes the tests performed to validate the SDK build in T5.3 (Implement TeamMate Extension SDK) in terms of usability and acceptability. The tests have been conducted with real developers who were asked to build a simple component/mobile app. The tests highlighted that the developers consider the SDKs quite usable and they would use them in case a new component or a mobile app for TeamMate is needed.

1 Introduction

In Automate project, the TeamMate system is coping with highly complex traffic situations. The TeamMate System, therefore, organizes and manages the automated functions according to the needs of the situation and the driver. The management of the functions are realized by advanced enablers, as the building blocks of the TeamMate System. As reported in previous deliverables (D5.1 and D5.4), the TeamMate system architecture is developed to specify the functional collaboration of enablers, the relations of the automate enablers among each other and together with a given platform (vehicle or simulator).

In this deliverable we focus on the extension of TeamMate system. We provide details for two complementary Extension SDKs: "TeamMate Component Extension SDK" and “Mobile app Extension SDK”.

These complementary SDKs are designed to enable the third party to extend both the built-in functionality (intrinsic behaviour) of TeamMate vehicle, as well as, the exterior functionality of the TeamMate vehicle.

The document also contains the description of the tests performed with real developers to validate the SDKs in terms of usability and acceptability.

This deliverable includes 7 chapters:

- Chapter 1: Introduction
- Chapter 2: Overview of the TeamMate Component Extension SDK
- Chapter 3: Overview of the Mobile App Extension SDK
- Chapter 4: TeamMate Component Extension SDK Deployment

- Chapter 5: Mobile app Extension SDK Deployment
- Chapter 6: Common Validation Framework
- Chapter 7: Conclusion

2 Overview of the TeamMate Component Extension SDK

The “TeamMate Component Extension SDK" provides the third parties with the tools to extend the intrinsic functionality of the TeamMate system. New enabler components can be adapted and integrated to the TeamMate system by only few steps of defining the massages and stablishing the required connections.

2.1 Component-based Structure

HMT developed a component-based Extension SDK to support the component-based architecture of TeamMate system. As explained in previous deliverables, the overall TeamMate functional architecture is based on several components called enablers. Multiple components collaborate with each other to process data, perform interpretations and plan the actions. Figure1 depicts the schematic architecture of TeamMate system described in detail in deliverable D5.1 and D5.4. The component-based architecture results in the configuration flexibility, in which the components are separated according to their concerns, allowing different setups to implement different interactions. To manage the interconnections of the components, a publisher-subscriber messaging pattern is introduced and used as application interface strategy. In this pattern, as explained in D5.1, each component may act as a server that provides services to other components. Simultaneously, each component may act as a client by requesting services from other components.

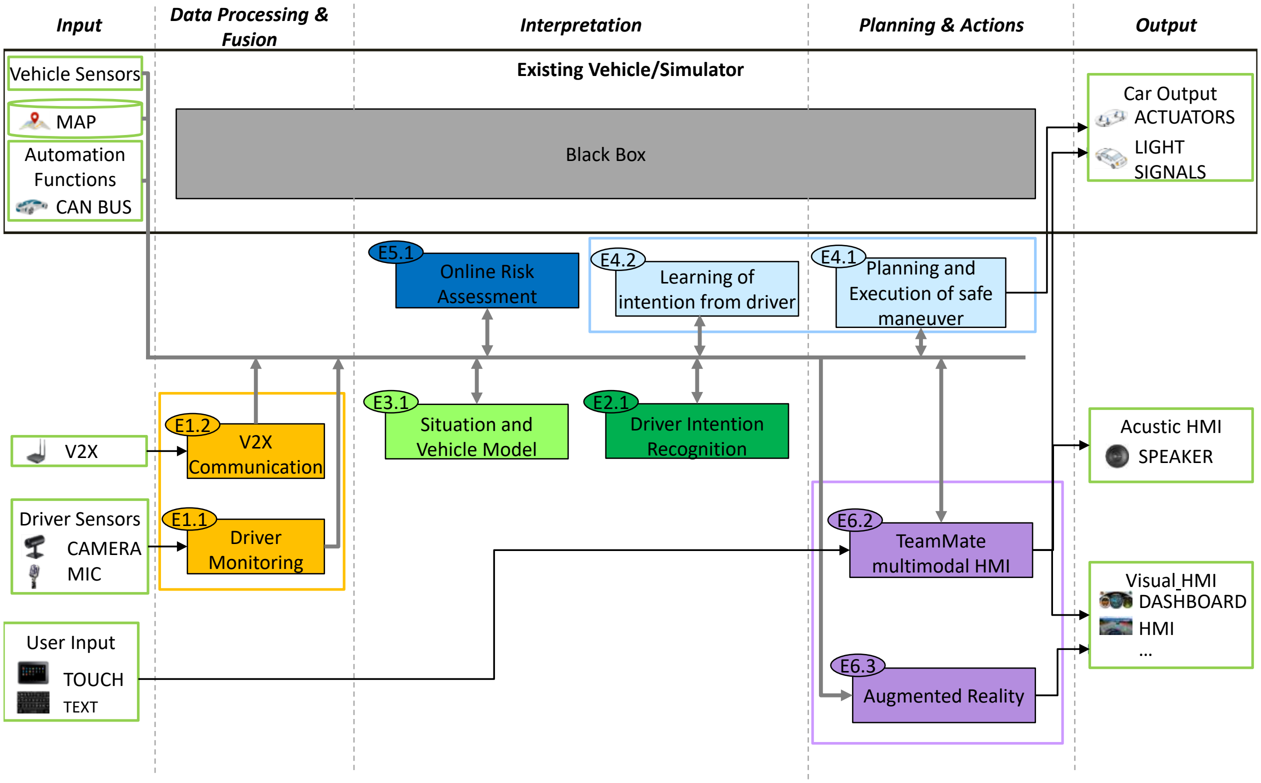


Figure 1: schematic diagram of the functional architecture of the TeamMate system.

To support the component-based architecture of TeamMate system and to establish the interconnection of the components, HMT developed a component-based Extension SDK with pre-built tools and libraries. Importantly, the extension SDK implements component-handlers, with the main functionality of managing the communication between the components using predefined publisher-subscriber messaging patterns. The component handles then binds to the enabler to build a component compound. This component compound is ready to be plugged in to the TeamMate system and to cooperate with other enablers, by establishing the connection, receiving input, producing output and sending it over the connection.

The developed SDK regulates the connection and integration of enablers to the Teammate system, by separating the component handler from enabler source code in a structured format. As a result, any enabler providers aiming at extending the TeamMate system (including third-party providers), only need to set up the component handler and bind it to the enabler. Setting up of the component handler could be done by using prebuilt tools of extension SDK and specifying the input/output message formats required by the enabler, the frequency of data transfer and the type of the connection to be established. Therefore, using the SDK, each enabler can be combined to a customized component handler. The combined enabler with component handler forms a TeamMate Component compound, which could directly connect to the automate system and replace previously connected components. To replace the enabler, simply remove the old enabler and plug the generated component compound to the system.

2.2 Component Compound

As an extension to the API support, provided by the first version of component-framework (see D5.1), HMT developed the component handler as a pre-implemented class to handle the interconnections and the input/output data transfer.

A component handler contains, all required information to establish the communication as well as send and receive data. A (third-party) developer can choose among pre-built communication and data transfer tools to customize the component handler. The customized component handler is then combined with an enabler to build a TeamMate component compound.

To combine the component handler with an enabler, two type of compositions are envisioned. In the first approach a standalone communication handler is called in the source code of the enabler. This approach is especially useful for integrating a single enabler to a TeamMate system. The communication handler establishes an application interface and establishes a separate thread for each of the data transfer stream. For example, it establishes a single thread for sending the output of the enabler over a network connection and repeatedly sends the provided output over. In a similar way it establishes threads for the input data stream and reads over a port with a specific frequency. The port numbers and data transfer frequency can be set in the source code of the resulting component compound (for more detailed explanation please see section 4.1.4).

In the second approach the enabler is embedded into a component handler, to produce a Teammate component compound. Then the TeamMate component compound, is embedded (called) in a component set. In the case where only a single enabler is getting integrated to TeamMate system, the component set contains a single component compound. In this approach, the component set is establishing the threads required for each of the components, and the inputs and outputs assigned for each of the components. Therefore, it provides a parallel execution of the enablers, while taking care of the communication between components and communication of components over network connections.

Like the first approach, this one also can be used to integrate a single component to the TeamMate system. However, this approach provides necessary tools to generate an extension package including multiple enablers. For this purpose, multiple component compounds, each including an enabler, are called in a single component set, generating a multi-component extension package with all required communication handlers.

As the next integration level, the TeamMate extension SDK, provides the option to integrate all enablers of a TeamMate system in a single set of components, where each component compound embeds an enabler. The enabler functions and communications between the components are handled internally by the component set, yet each component can establish a connection over network to any other enabler provider. The component set class creates separate threads for enabler executers and for inputs and outputs associated to each of the component handlers. Therefore, the execution of multiple components can be done in parallel, while the connections are established and handled in separate threads with specified frequencies in configuration files (section 4.1.4).

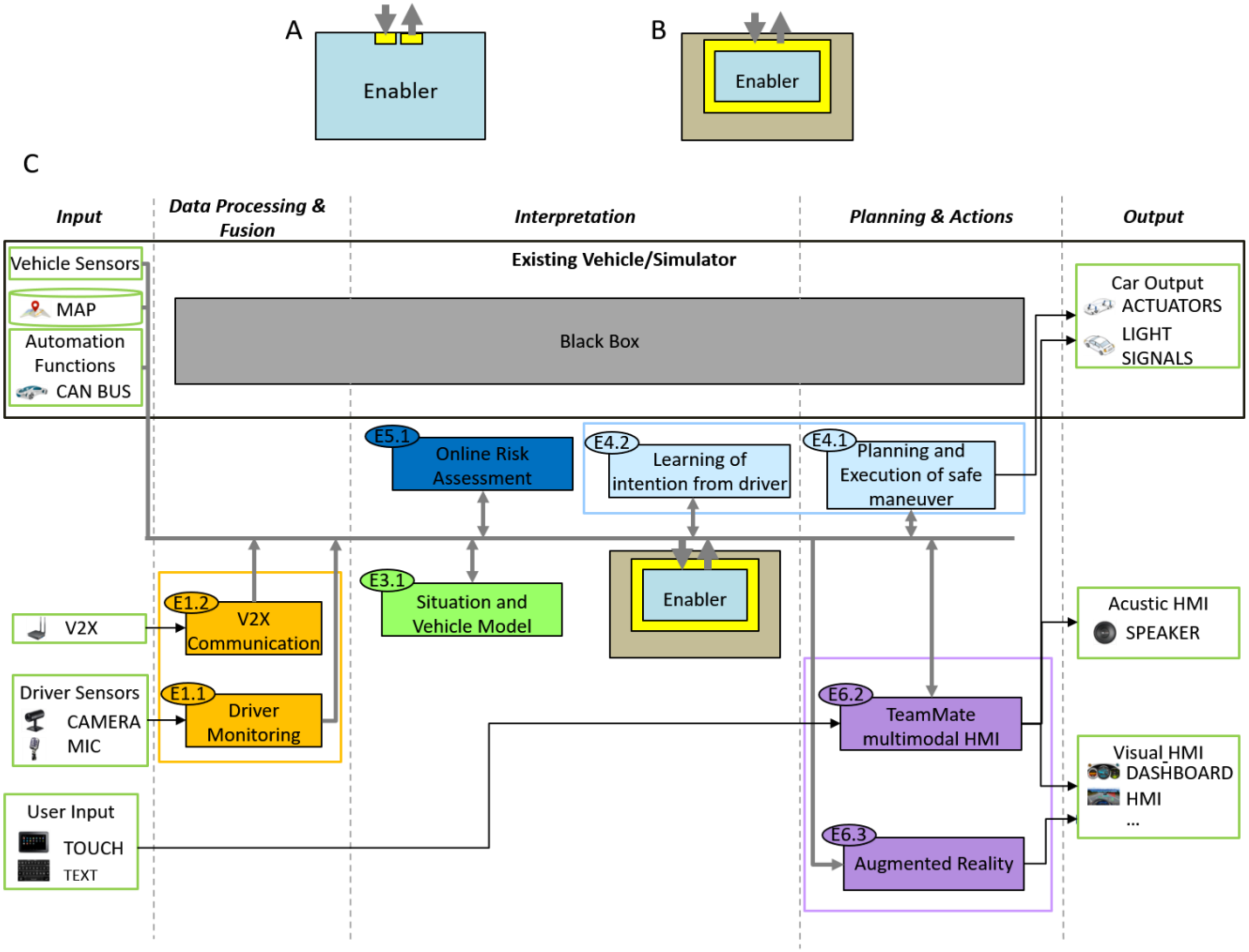


Figure 2: Schematic diagram of embedding an enabler into a handler/connector component to establish a TeamMate component with build-in communication functionalities.

Figure 2 depicts schematically the preparation of a component compounds using two composition types. Figure 2A depicts the embedding of component handler, including standalone input and output in an enabler source code. In Figure 2B, the enabler is called from the execute method of a component handler, to embed the enabler in a component. The component is then placed in a component set (brown), to form the final executable TeamMate component compound.

Each of the enablers in the TeamMate architecture can be replaced by an equivalent version of TeamMate component compound. As an example, in Figure 2C, the Enabler 2.1 in Figure 1 is replaced by the component compound from Figure 2B.

2.3 Overview of SDK Deployment

The SDK files contain the source files written in C++ and a user manual, which explains the steps to install and deploy the package.

The source files include the tools and libraries required to develop component compounds. It also contains several component compounds code examples, which covers different tools and options available to develop components. These examples could be used as template components and, therefore, the third-party can develop their own components by the modification of the template component on the commented sections, for example by defining the input/output data structures and replacing their source code in the section marked with comments.

HMT developed the SDK to be used by any chosen compiler environment. To this aim, the IDE specific project files are generated using CMake, which is an open-source, cross-platform family of tools designed to build software. This implementation enables the developers to build the component compounds using any C++ IDEs of own choice.

TeamMate Extension SDK utilize Boost cross-platform library for network and I/O programming. This library is used for a consistent asynchronous model using a modern C++ approach.

All the prerequisites and source code to build the library are placed in the software package. The step-by-step installation of the prerequisites and generation of the project files using the source code is explained in the user manual (Appendix 1: User Manual and Source Code for TeamMate Extension SDK).

3 Overview of the Mobile App Extension SDK

Thanks to the installation of an IoT embedded device developed by REL (from now on called "Gino‟), the real-time information about the vehicle (available on the CAN bus) have been extracted, elaborated and made available to easily create third-party mobile app.

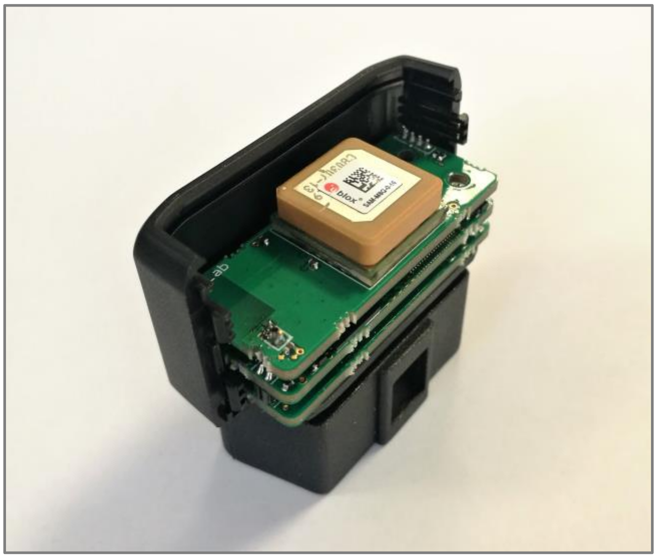


Figure 3: "Gino" IoT embedded device to be connected to the vehicle

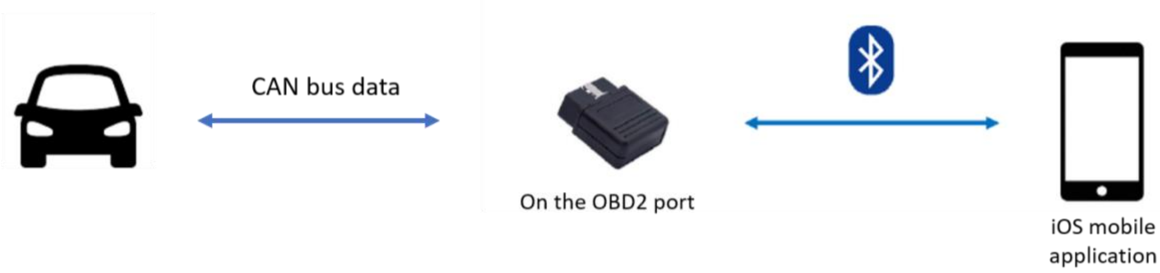


Figure 4: Overall architecture of the Mobile APP Extension SDK

The SDKs developed in the Automate project have been used to seamlessly create new mobile applications that exploit this data.

An example of a potential application is the re-use of driving data by an insurance company. In fact, it could use the real-time information to identify the driving behavior and then associate the risk of the driver (to optimize the driver's profile). These systems are already available on the market and could be further improved by the introduction of the Automate SDK.

This approach is in line with the EU strategy about the re-use of available data to strengthen the EU economy and creation of an innovation ecosystem.

3.1 Architecture

The general architecture of "Gino" is composed by two main parts (as shown in Figure 5):

- "Gino" Library
- "Gino" SDK (for iOS / Android)

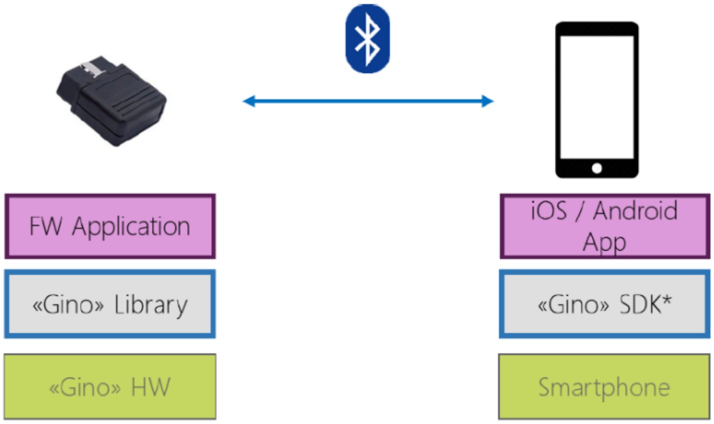


Figure 5: Firmware and software architecture of the Mobile App Extension SDK

The “Gino” SDK can be used by the iOS/Android application in order to communicate to the OBD hardware and read CAN signals. The “Gino” Library is responsible to provide to the application all the APIs needed to communicate to the “Gino” SDK through the DQ technology.

The detailed description of APIs to use the SDKs is provided in Appendix 2: User Manual for ThirdParty APP/HMI Extension SDK.

3.2 “Gino” Library

The “Gino” Library is composed by the following SW modules (as shown in Figure 6):

- DQuid stack
- Minimal Control Function (MCF) stack
- BLE stack
- HW stack

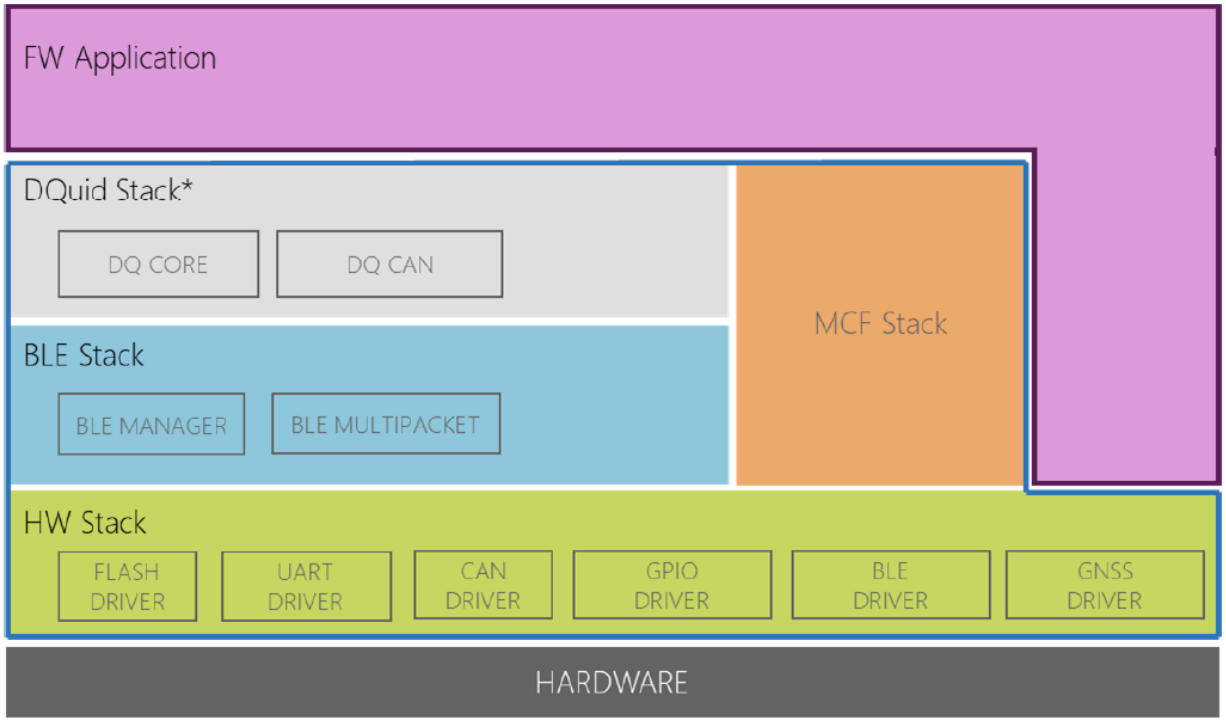


Figure 6: Software modules of the "Gino" Library

3.2.1 DQuid Stack

Static C library - DQuid Stack is built for the target system (e.g. FreeRthos, Linux, QnX, ..), including targets w/o OS:

- Object identity secure storing, object authentication
- Exchange/update any data sourced by peripherals (I/O, CAN, Serial, etc.) in form of signals (DQSignals) toward the DQuid Protocol to DQuid SDK
- Data security: AES 256 encryption algorithm, asymmetric key generation
- Portability: hardware independent. The Stack does not depend to hardware peripherals so that can be easily ported in any platform
- Interface Integration and set up towards framework and APIs of IoT systems

3.2.2 MCF Stack

The MCF Stack, includes the functionalities of the Minimal Control Function chapter of the ISOBUS standard. This chapter includes the SAE J1939 and multi packet message management.

REL customized the ISOBUS software stack that covers the following SAE J1939 functionalities:

- Address Claim management (ISO11783-5)
- Receive/Transmit single packet messages
- Receive/Transmit multi packet messages (TP/BAM protocol, ETP non supported)

3.2.3 BLE Stack

This software module is responsible of BT management:

- BLE initialization
- BLE connection state machine
- BLE advertising packet
- Data send / receive on BLE channel

The BLE stack use the developed low level BLE driver module provided by the HW module.

3.2.4 HW Stack

The HW stack module includes the following software components:

- Low level CAN API
- External Flash API
- Power Management API
- GPIO API (to control LED)
- UART driver
- BLE driver (The BLE integrate a BGLibTM host library which implements BGAPI protocol)

3.3 "Gino" SDK

The Gino SDKs is an iOS static library/framework (supporting iOS >= 10.0) and an Android jar (supporting OS >= 5.0).
The "Gino" SDK in build on top of DQuid SDK to enable the communication with "Gino" device through the DQuid technology.

- Simplicity of use: an app developer is not asked to fully know the language either of the object or of any IoT framework: we set up a single interface enabling interaction with the object according to the following paradigm: Object. Property = Value;
- Secure data exchange with DQuid Stack toward the DQuid Protocol.
- Connected Object configuration: the object (e.g. a Car) embedding the DQuid stack becomes a DQuidObject with properties (e.g. speed, RPM), private or public.
- Connected object data access: data are object's properties, accessible in r/w as car.speed, car.rpm, etc.

- Authentication, authorization: toward DQuid Web APIs, all stakeholders are authenticated/authorized (end user, developer, app and object)

3.3.1 DQObject – DQSignal

"DQuid protocol" is the data exchange protocol used by the DQuid SDK - integrated into the mobile application - and DQuid Stack - integrated into the application running on the embedded device.

The mobile application (Android / iOS) links the "DQuid SDK" framework that is in charge of establishing a communication with the embedded device on a specific link (e.g. BLE) and exchange data over this link.

The embedded device integrating the DQuid Stack is represented in the DQuid SDK as a DQuidObject with one or more properties (DQSignal).

The mobile application can read/write the DQuidObject's properties.

The mobile application can subscribe to all DQuidObject's properties in order to receive notifications when properties data are updated by the embedded device.

DQuidObject's properties are defined by the mobile application developer toward the DQuid SDK; properties are stored in JSON format.

3.3.2 DQuid SDK – Core Module

DQuid SDK provides the following features:

- Discovery of the devices embedding the DQuid Stack
- Connection/Disconnection to/from a specific embedded device (one connection at a time)

- Connection/Disconnection notification
- Definition of object's properties (DQSignal) with attributes (size, type, readable/writable). These properties must also be specified in the embedded firmware application integrating the DQuid Stack.
- Properties' read/write
- Properties' Subscribe/Unsubscribe
- Properties' update notification (in case of property subscription).

3.3.3 DQuid SDK – CAN Module

- DQuid SDK provides a module for the CAN signal properties management.
- Every CAN signal in a CAN message is translated into a DQSignal property.
- DQSignals properties of type CAN have additional attributes to store specific CAN signal data needed to source signal's information from within the CAN message, specifically:
  - Start Bit
  - Length
  - CAN message ID
  - CAN channel ID
- DQuid SDK provides a method to parse the CAN db (Vector format .dbc) and automatically populate the properties' JSON file with all CAN signals and messages stored into the DBC.

DQuid SDK - CAN Module provides the following features:

- .dbc database parsing and automatic creation of the DQSignal properties with all attributes (Start Bit, Length, CAN message ID, CAN channel ID).

- Subscription/Unsubscription of CAN signals (properties' names are in the form «CANMessage.CANSignal»)
- For each CAN signal, it allows to define the transmit rate of each property from the DQuid Stack to the DQuid SDK.
- For each CAN signal, it allows to define the way the signal is elaborated between consecutive BLE transmissions (LAS, AVERAGE, MIN, MAX). This elaboration is allowed only for 32bits unsigned signals.
- Subscription/Unsubscription of CAN messages (properties names are in the form «CANMessage»)
- For each CAN message, it allows to define the transmit rate of each property from the DQuid Stack to the DQuid SDK.
- CAN signal/message update notification for all subscribed signals. The DQuid SDK applies the proper offset and scale factor to provide the CAN signal physical value to the mobile application.
- CAN message writing.
- Optionally, it allows to define the CAN message transmit rate (on the CAN network)
- It allows to update the payload of the CAN message transmitted over the CAN network.

4 TeamMate Component Extension SDK Deployment

The Extension SDK pre-implemented functions to build a custom component handler for a desired enabler. Each component contains an “execute” method, which is called continuously while the component is running. The execute part is responsible to read inputs, perform all calculations required to produce the output and to write outputs. Here, it is the place which the enabler owner needs to read inputs, to pass it to the enabler and to extract output from enabler, and finally to write the output.

Having an enabler in hand, the following objects and parameters needs to be specified in the component handler to form a component compound:

4.1.1 Inputs and Output

To be able to establish the connection, the input and output classes are defined which are responsible to handle receiving the input data (read) and sending the output data (write).

Each component can receive multiple inputs. It only provides one output.

For each input and output, a separate thread is established, to be able to read and write with a desired frequency independently of the time required for execution of the other inputs and outputs, or the required time for the processing of the execute method. Input and output classes contained pre-implemented read and write functions, which will be called for different serialization types and connection protocols.

4.1.2 Serialization

The SDK supports three types of serialization: Protobuf, boost and Qt.

As mentioned in the previous section, inputs and outputs objects call the pre-implemented read/write functions based on the selected serialization types. In Automate, most of the data structures for the communication of enablers are currently defined following the protobuf structure (as. proto files). However, in Automate, each partner uses a developing platform with predefined communication types based on the present simulator or vehicle and therefore the SDK needs to be able to handle the required communication types. For example, the simulator software of the REL simulator restricts the communication to use QT serialization. Therefore HMT, extended the TeamMate extension SDK to support QT message transfer. As another frequently used serialization type, boost serialization is also supported by SDK. The boost serialization has not been used by any of the Automate partners, yet, a third party could use this serialization type, if needed.

4.1.3 Connection Protocol

The SDK supports both commonly used communication protocols: UDP and TCP. On Automate most of the connections are established over TCP networks.

4.1.4 Configuration File

The configuration files (.ini files) specify all the required information by inputs and outputs object to establish the connection. For each input or output object a configuration file needs to be specified. It should be placed in the same folder as the component .exe file and can be modified at any time independently.

The configuration file contains the port numbers, host address and the frequency of the data transfer. The port numbers are used to connect the desired inputs to the chosen outputs. In case the component exe file is placed in a second computer, it uses the host address to find the desired computer and to connect over, e.g., internet connection. The frequency is the other important parameter, defined in the configuration file and defines the frequency of data transmission. Data arrived in higher frequency than specified, will be discarded in part based on the frequency difference.

4.2 Example Component Development

We used the TeamMate extension SDK to build three different TeamMate components. The Driver Intention Recognition (DIR) component was built and used to extend the REL TeamMate system by integrating the DIR enabler in entering to the roundabouts. The Online learning component was developed as part of a SiLab Data Processing Unit (DPU) and extends the SiLab simulator TeamMate system. Finally, the HMI Augmented reality with Epson glasses benefits from the SDK for its integration with the DIR RTmaps component in the VED vehicle.

For combining the component handler with enablers, we used both integration options, offered by SDK:

1- Embedding the enabler in a component handler,

2- Embedding a standalone communication handler into the enabler.

We used the first option to build the driver intention recognition component. The second option is used for building the online learning component and for the integration of the augmented reality with the Epson glasses.

4.2.1 Driver Intention Recognition Component

We used the extension SDK to extend the REL TeamMate system by the intention recognition enabler (Enabler 2.1). The intention recognition enabler consumes the ego vehicle data and object data from the simulator and calculates the probability of driver intention in entering roundabouts, whether the driver intends to enter to the roundabout or wait at the entrance of the roundabout. The intention recognition enabler was not integrated in the REL system before. Therefore, here, the TeamMate extension SDK used to extend the REL TeamMate System.

As discussed in section 2, we first build a TeamMate component by combining a component handler with the intention recognition enabler. To combine the component handler with the enabler, we embedded the enabler in a component handler scheme (deployment of solution 1, Figure 2B). The resulted TeamMate component is compiled to an executable file, which was used directly on the simulator computer.

By running the exe-file on the computer, the intention recognition component establishes a connection to REL simulator and handles receiving the data from simulator and sending the intention of the driver as the output. The execute function embedded in this component performs all required calculations to produce the output of the component, which is the intention probability of the driver to enter to the roundabout.

UDP connection is used to establish the connection between the REL simulator and the intention recognition component compound. The data is serialized and de-serialized using QT serialization. To this aim, the data structures, depicted in Tables 1 and 2 are used to stream inputs to the intention recognition component and to send the output of the intention recognition component to the simulator.

Table 1: Input Class of Driver Intention Recognition to Enter to the Roundabout.

```
#pragma once
#include <QtCore>
struct QTEgoAlterVehicle
{
public:
        QTEgoAlterVehicle() :
                timestamp(),
                Ego_Road_Id(),
                Ego_Abscissa(),
                Alter_3_Road_Id(),
                Alter_3_Lane_Id(),
                Alter_3_Abscissa(),
                Alter_3_Speed(),
                Alter_4_Road_Id(),
                Alter_4_Lane_Id(),
                Alter_4_Abscissa(),
                Alter_4_Speed(),
                Alter_0_Road_Id(),
                Alter_0_Lane_Id(),
                Alter_0_Abscissa(),
                Alter_0_Speed()
        {}

        float timestamp;
        qint64 Ego_Road_Id;
        float Ego_Abscissa;
        qint64 Alter_3_Road_Id;
        qint64 Alter_3_Lane_Id;
        float Alter_3_Abscissa;
        float Alter_3_Speed;
        qint64 Alter_4_Road_Id;
        qint64 Alter_4_Lane_Id;
        float Alter_4_Abscissa;
        float Alter_4_Speed;
        qint64 Alter_0_Road_Id;
        qint64 Alter_0_Lane_Id;
        float Alter_0_Abscissa;
        float Alter_0_Speed;
```

```
    void fromByteArray(QByteArray datagram)
    {

        QDataStream dataStream(&datagram, QIODevice::ReadOnly);
        dataStream.setVersion(QDataStream::Qt_5_3);
        dataStream
            >> timestamp
            >> Ego_Road_Id
            >> Ego_Abscissa
            >> Alter_3_Road_Id
            >> Alter_3_Lane_Id
            >> Alter_3_Abscissa
            >> Alter_3_Speed
            >> Alter_4_Road_Id
            >> Alter_4_Lane_Id
            >> Alter_4_Abscissa
            >> Alter_4_Speed
            >> Alter_0_Road_Id
            >> Alter_0_Lane_Id
            >> Alter_0_Abscissa
            >> Alter_0_Speed;

    }
    QByteArray toByteArray()
    {
        QByteArray datagram;
        QDataStream out(&datagram, QIODevice::WriteOnly);
        out.setVersion(QDataStream::Qt_5_3);
        out << timestamp
            << Ego_Road_Id
            << Ego_Abscissa
            << Alter_3_Road_Id
            << Alter_3_Lane_Id
            << Alter_3_Abscissa
            << Alter_3_Speed
            << Alter_4_Road_Id
            << Alter_4_Lane_Id
            << Alter_4_Abscissa
            << Alter_4_Speed
            << Alter_0_Road_Id
            << Alter_0_Lane_Id
            << Alter_0_Abscissa
            << Alter_0_Speed;
        return datagram;
    }
};
```

Table 2: Output Class of Driver Intention Recognition to Enter to the Roundabout.

```
#pragma once
#include <QTCore>
struct QTIntention
{
        float time;
        qint32 intention;
        QTIntention() :
               time(0.0f),
               intention(0)
        {}
        QByteArray toByteArray()
        {
               QByteArray datagram;

               QDataStream out(&datagram, QIODevice::WriteOnly);
               out.setVersion(QDataStream::Qt_5_3);
               out << time<< intention;
               return datagram;
        }
        void fromByteArray(QByteArray datagram)
        {
               QDataStream dataStream(&datagram, QIODevice::ReadOnly);
               dataStream.setVersion(QDataStream::Qt_5_3);
               dataStream
                      >> time
                      >> intention;
        }
};
```

4.2.2 Online Learning Component

The Online Learning Component is basically the Enabler 4.2 "Learning of Intention from the driver" extended by the possibility to communicate with its Online Learning Visualization application. For the SiLab simulator TeamMate system, as it is used at ULM, the enabler E4.2 is part of a SiLab DPU. To be able to monitor the updating of the DIR model while driving in the simulation environment, a standalone communication handler from the SDK is embedded in the implemented enabler E4.2 (deployment solution 2 as illustrated in Figure 2A). A protocol buffer message was defined, as shown in Table 3, to serialize the relevant DIR model parameters and send them to the Online Learning Visualization. In contrast to the previous example, the message is not sent at a fixed frequency but only on demand whenever the DIR model was updated. This is implemented by passing the message to the *write()* method of an instance of the *AM::SA_TCPOutput* class. This is shown in Table 4. By feeding the data which is required for the visualization to a stand-alone application, the visualization may run on a separated machine and does therefore not consume resources on the machine where the actual online learning takes place.

Table 3: Online Learning Visualization message

```
syntax = "proto2";
package eu.automate.openapi.messages;

message OnlineLearningOutputMessage {
  optional int64 timestamp          = 1; // Timestamp
  optional uint32 initial           = 2; // 1 if contains inital model.
  message DistributionGauss  {
    required string model_name      = 1; // Name of the model
    repeated string child_names     = 2; // Names of the child variables
    repeated string parent_names    = 3; // Names of the parent variables
    repeated DataGauss data         = 4;
  }
  message DataGauss {
    repeated int32 parent_states    = 1;   // States of the parent variables
    required float mean             = 2;   // mean of the gaussian
    required float var              = 3;   // variance of the gaussian
  }
  message DistributionDiscrete  {
    required string model_name           = 1; // Name of the model
    required string variable_name        = 2; // Name of variable
    repeated string parent_names         = 3; // Names of the parent variables
    required uint32 variable_cardinality = 4; // Cardinality of the variable
    repeated DataDiscrete data           = 5;
  }
  message DataDiscrete {
    repeated int32 parent_states    = 1; // States of the parent variables
    repeated float probabilities    = 2; // Vector of probabilities, with one
probability of each possible assignment of the discrete variable
  }
  message DistributionMog {
    required string model_name   = 1; // Name of the model
    repeated string parent_names = 2; // Names of the parent variables
    repeated string child_names  = 3; // Names of the child variables
    repeated DataMog data        = 4;
  }
  message DataMog {
    repeated int32 parent_states = 1; // States of the parent variables
    repeated float means         = 2; // means of the gaussian components
    repeated float vars          = 3; // variances of the gaussian components
    repeated float weights       = 4; // weights of the gaussian components
  }
  repeated DistributionDiscrete discretes = 3; // Vector of discrete distribu-
tions
  repeated DistributionGauss gaussians    = 4; // Vector of gaussian distribu-
tions
  repeated DistributionMog mogs           = 5; // Vector of MoG distributions
}
```

Table 4: Schematic sample code for the use of a standalone communication handler

```
// start communication socket for OL visualization at port 1000
sender = new
AM::SA_TCPOutput<AM::ProtobufSerialization<eu::automate::openapi::messages::OnlineLearn
ingVisualizationMessage>,eu::automate::openapi::messages::OnlineLearningVisualizationMe
ssage>(1000);

while (receiving_data){
  updated = false;
  perform_online_learning();
  // If the model was updated recently write the data for distribution visualization
  if (updated){
    eu::automate::openapi::messages::OnlineLearningVisualizationMessage msg;
    // write model parameter serialization to message object
    distribution_visualization::make_pb_message(model, msg);
    auto msg_ptr = std::make_shared<
                eu::automate::openapi::messages::
                OnlineLearningVisualizationMessage>(msg);
    sender->write(msg_ptr);
  }
}
```

4.2.3 Component Functionality Extension

Initially the Augmented Reality (AR) HMI was only planned for the simulator demonstrators. However, to realize some similar HMI also in a real car demonstrator, a solution based on AR glasses was implemented. The RTmaps component that is used to integrate the DIR, the Online Risk assessment, and other enablers into the VED vehicle was extended to produce a protocol buffer message which provides necessary data to trigger certain states of the AR HMI. Additionally, an Android application was developed, which is deployed to the Epson glasses to receive the message and to control the HMI on the glasses. The structure of the message is shown in Table 5. More details about this HMI version and the related Android App can be found in deliverable D6.3

Table 5: AR Glasses HMI message to send relevant data for the triggering of HMI states to the Epson glasses

```
syntax = "proto2";
package eu.automate.openapi.messages;

message ArHmiMessage {
    required int64 timestamp                = 1;   // Timestamp
    required int32 driver_intention         = 2;   // current driver intention
    required int32 trajectory_risk          = 3;   // trajectory risk (0: not assessed, 1: safe, 2: unsafe)
    required int32 ego_lane                 = 4;   // current ego lane
    required int32 thw_thresh               = 5;   // 1 if below TimeHeadway threshold
}
```

5 Mobile app Extension SDK Deployment

"Mobile app Extension SDK" provides access to the CAN bus of a vehicle to easily create mobile apps that use this data. Therefore, this SDK enables third parties to develop new mobile applications that exploit the potential of the functionality of the TeamMate system without having a deep knowledge of the hardware and firmware of the vehicle.

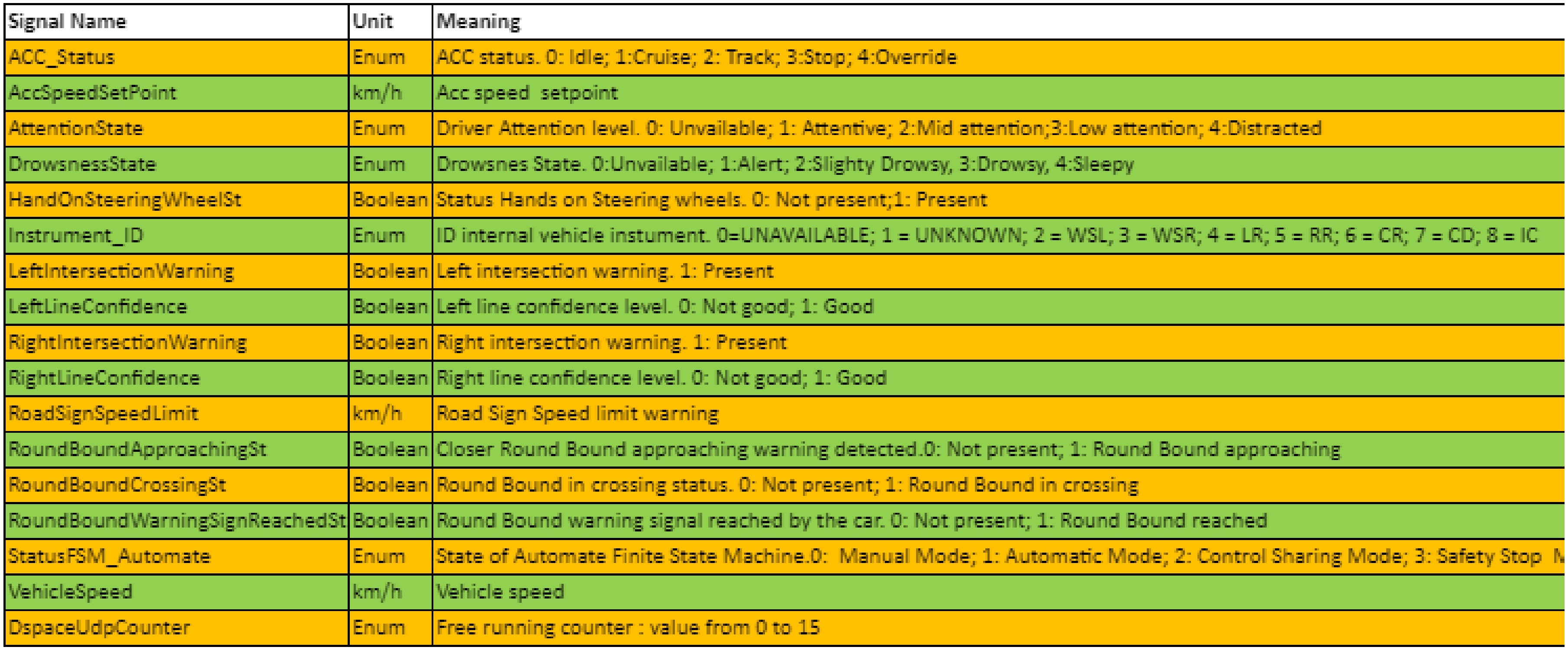
5.1 Distributed HMI

The application was aimed at defining a Proof of Concept to verify (and potentially demonstrate) that, if the vehicle is equipped with a system able to recognize the visual Area of Interest and provide info on a specific display (e.g. through a device able to read/write data on the CAN network), the reaction time for the take over can be reduced, potentially increasing the safety.

In this sense, the application should warn the driver to suggest a (partial or total) resumption of control from automated driving. The conceptual conditions that activate the transition of control are:

1. The car performs a (partial or total) take over request
2. The driver is looking at the specific Area of Interest (AoI) in which the smartphone is placed (this information is collected by the DMS and communicated on the CAN bus in real time)

CRF has provided its database with the CAN bus signals used for the communication with the mobile app.



Signal Name	Unit	Meaning
ACC_Status	Enum	ACC status. 0: Idle; 1:Cruise; 2: Track; 3:Stop; 4:Override
AccSpeedSetPoint	km/h	Acc speed setpoint
AttentionState	Enum	Driver Attention level. 0: Unvailable; 1: Attentive; 2:Mid attention;3:Low attention; 4:Distracted
DrowsnessState	Enum	Drowsnes State. 0:Unvailable; 1:Alert; 2:Slighty Drowsy, 3:Drowsy, 4:Sleepy
HandOnSteeringWheelSt	Boolean	Status Hands on Steering wheels. 0: Not present;1: Present
Instrument_ID	Enum	ID internal vehicle instument. 0=UNAVAILABLE; 1 = UNKNOWN; 2 = WSL; 3 = WSR; 4 = LR; 5 = RR; 6 = CR; 7 = CD; 8 = IC
LeftIntersectionWarning	Boolean	Left intersection warning. 1: Present
LeftLineConfidence	Boolean	Left line confidence level. 0: Not good; 1: Good
RightIntersectionWarning	Boolean	Right intersection warning. 1: Present
RightLineConfidence	Boolean	Right line confidence level. 0: Not good; 1: Good
RoadSignSpeedLimit	km/h	Road Sign Speed limit warning
RoundBoundApproachingSt	Boolean	Closer Round Bound approaching warning detected.0: Not present; 1: Round Bound approaching
RoundBoundCrossingSt	Boolean	Round Bound in crossing status. 0: Not present; 1: Round Bound in crossing
RoundBoundWarningSignReachedSt	Boolean	Round Bound warning signal reached by the car. 0: Not present; 1: Round Bound reached
StatusFSM_Automate	Enum	State of Automate Finite State Machine.0: Manual Mode; 1: Automatic Mode; 2: Control Sharing Mode; 3: Safety Stop M
VehicleSpeed	km/h	Vehicle speed
DspaceUdpCounter	Enum	Free running counter : value from 0 to 15

Figure 7: Signals of the CRF CAN bus

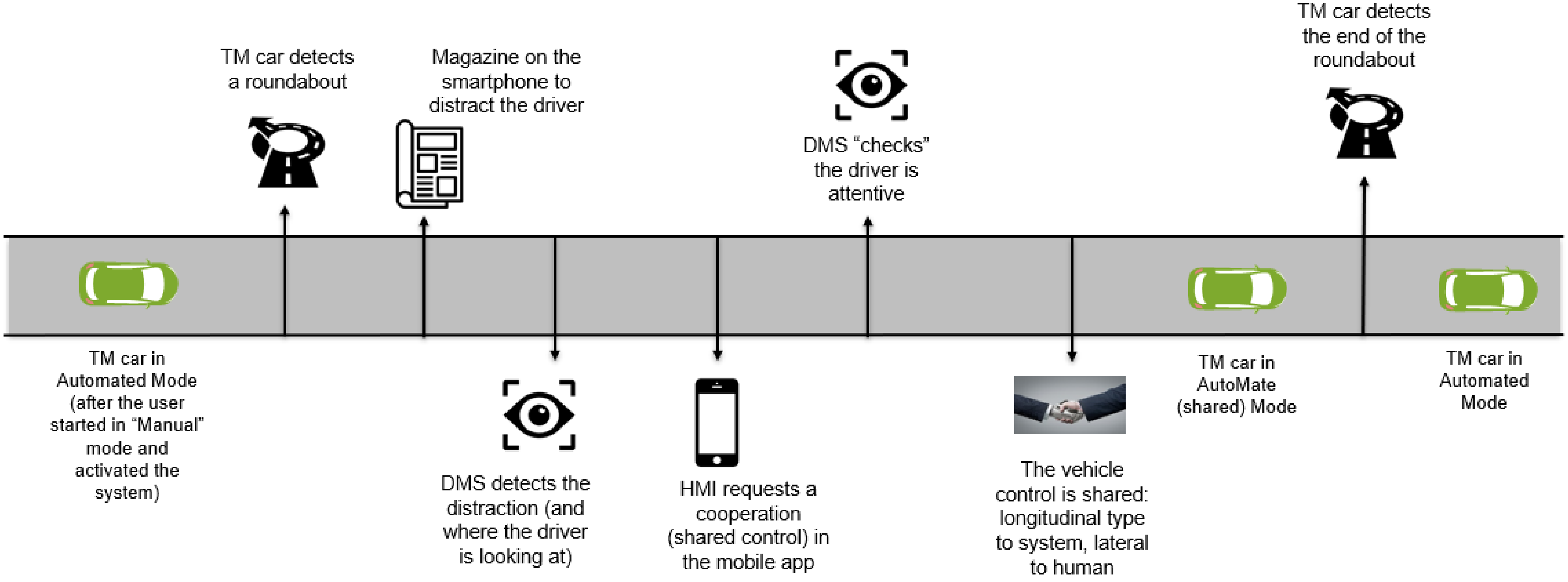
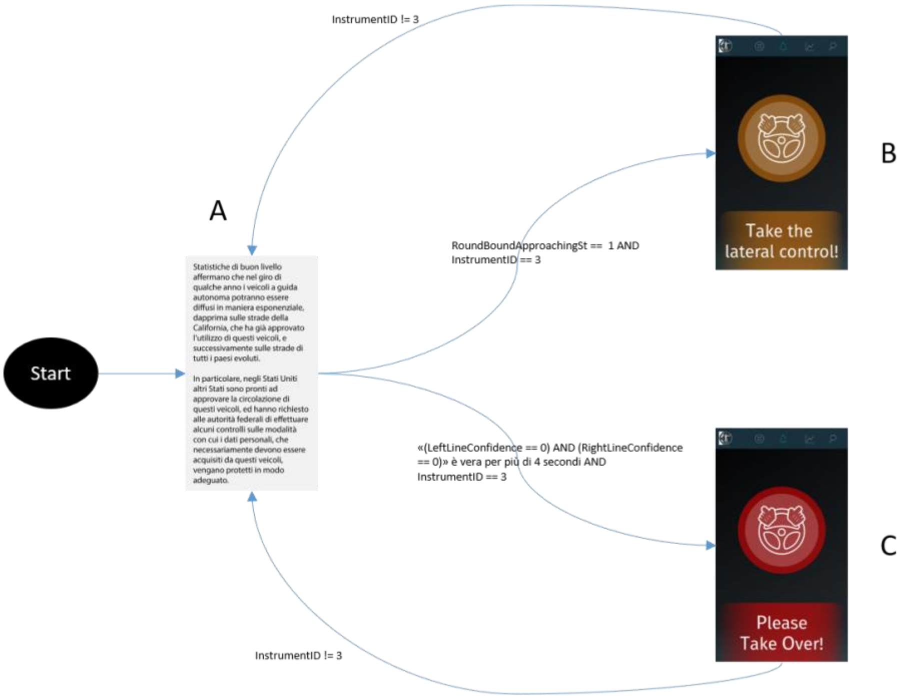

TM car detects a roundabout
Magazine on the smartphone to distract the driver
DMS "checks" the driver is attentive
TM car detects the end of the roundabout
TM car in Automated Mode (after the user started in "Manual" mode and activated the system)
DMS detects the distraction (and where the driver is looking at)
HMI requests a cooperation (shared control) in the mobile app
The vehicle control is shared: longitudinal type to system, lateral to human
TM car in AutoMate (shared) Mode
TM car in Automated Mode

Figure 8: EVA scenario where the mobile app has been implemented and integrated

The mobile app is based on the EVA scenario (represented in Figure 8):

- The car is driving in Automated Mode
- The experimenter activates the smartphone and tell the user to read aloud the text in the application
- The experimenter places the smartphone in a central position, in the lower part of the central tunnel (AoI number 3 of the DMS)
- When the car detects a roundabout (RoundBoundApproachingSt == 1), a request to take the lateral control is activated (TLCR). Since the driver is watching the smartphone (the driver is watching in the area in which the vehicle expects a smartphone is placed and the DMS recognize it), the warning is given on the smartphone, as a pop-up info consistent with the HMI.
- The driver takes the lateral control and passes the roundabout in “Shared Control” Mode; in this phase the application returns in the homepage (i.e. the screen “A”) and the experimenter keep the smartphone out of the original area; then s/he shuts down the application.


InstrumentID != 3
A
B
Take the lateral control!
RoundBoundApproachingSt == 1 AND InstrumentID == 3
Start
«(LeftLineConfidence == 0) AND (RightLineConfidence == 0)» è vera per più di 4 secondi AND InstrumentID == 3
C
Please Take Over!
InstrumentID != 3

Statistiche di buon livello affermano che nel giro di qualche anno i veicoli a guida autonoma potranno essere diffusi in maniera esponenziale, dapprima sulle strade della California, che ha già approvato l'utilizzo di questi veicoli, e successivamente sulle strade di tutti i paesi evoluti.

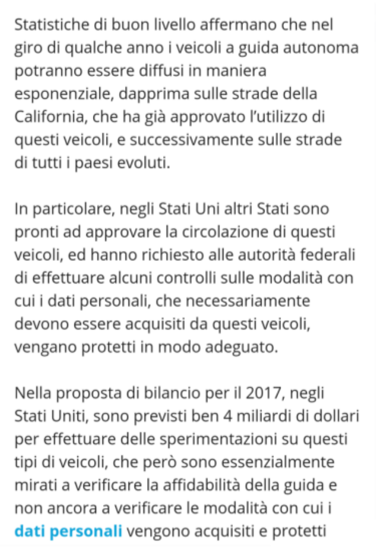
In particolare, negli Stati Uniti altri Stati sono pronti ad approvare la circolazione di questi veicoli, ed hanno richiesto alle autorità federali di effettuare alcuni controlli sulle modalità con cui i dati personali, che necessariamente devono essere acquisiti da questi veicoli, vengano protetti in modo adeguato.

Figure 9: State machine of the mobile app

As shown in the state machine (Figure 9), 3 layouts have been created to implement the applications:

- A screenshot with a plain text (used to visually distract the driver)
- A request to take the lateral control
- A request to take over (lateral and longitudinal control)

The app has been used to evaluate the performance of the TeamMate system integrated into the CRF vehicle, and showed the positive impact of this HMI distributed concept on safety.

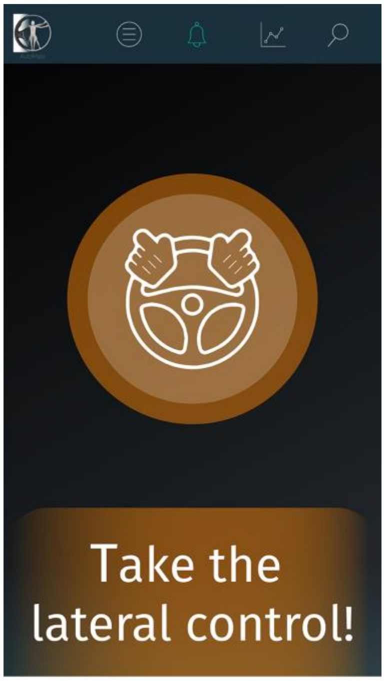


Statistiche di buon livello affermano che nel giro di qualche anno i veicoli a guida autonoma potranno essere diffusi in maniera esponenziale, dapprima sulle strade della California, che ha già approvato l'utilizzo di questi veicoli, e successivamente sulle strade di tutti i paesi evoluti.

In particolare, negli Stati Uni altri Stati sono pronti ad approvare la circolazione di questi veicoli, ed hanno richiesto alle autorità federali di effettuare alcuni controlli sulle modalità con cui i dati personali, che necessariamente devono essere acquisiti da questi veicoli, vengano protetti in modo adeguato.

Nella proposta di bilancio per il 2017, negli Stati Uniti, sono previsti ben 4 miliardi di dollari per effettuare delle sperimentazioni su questi tipi di veicoli, che però sono essenzialmente mirati a verificare la affidabilità della guida e non ancora a verificare le modalità con cui i **dati personali** vengono acquisiti e protetti

Figure 10: Initial screenshot with a plain text to distract the driver



Take the lateral control!

Figure 11: Request to take the lateral control

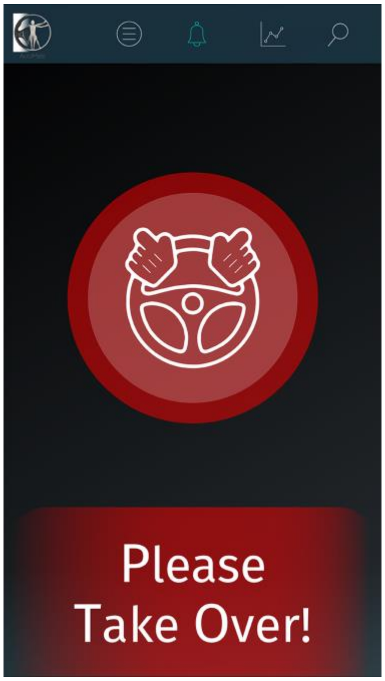


Figure 12: Request to take over

6 Common Validation Framework

6.1 Objective of the Validation

A common framework has been defined to validate both SDKs in terms of usability and acceptability.

6.2 Experimental design

A common experimental design has been defined for the validation of both SDKs:

1) Selection of participants (i.e. developers)

2) Definition of tasks to be completed in order to properly test the SDKs

3) Administration of usability and acceptability questionnaires (SUS and TAM)

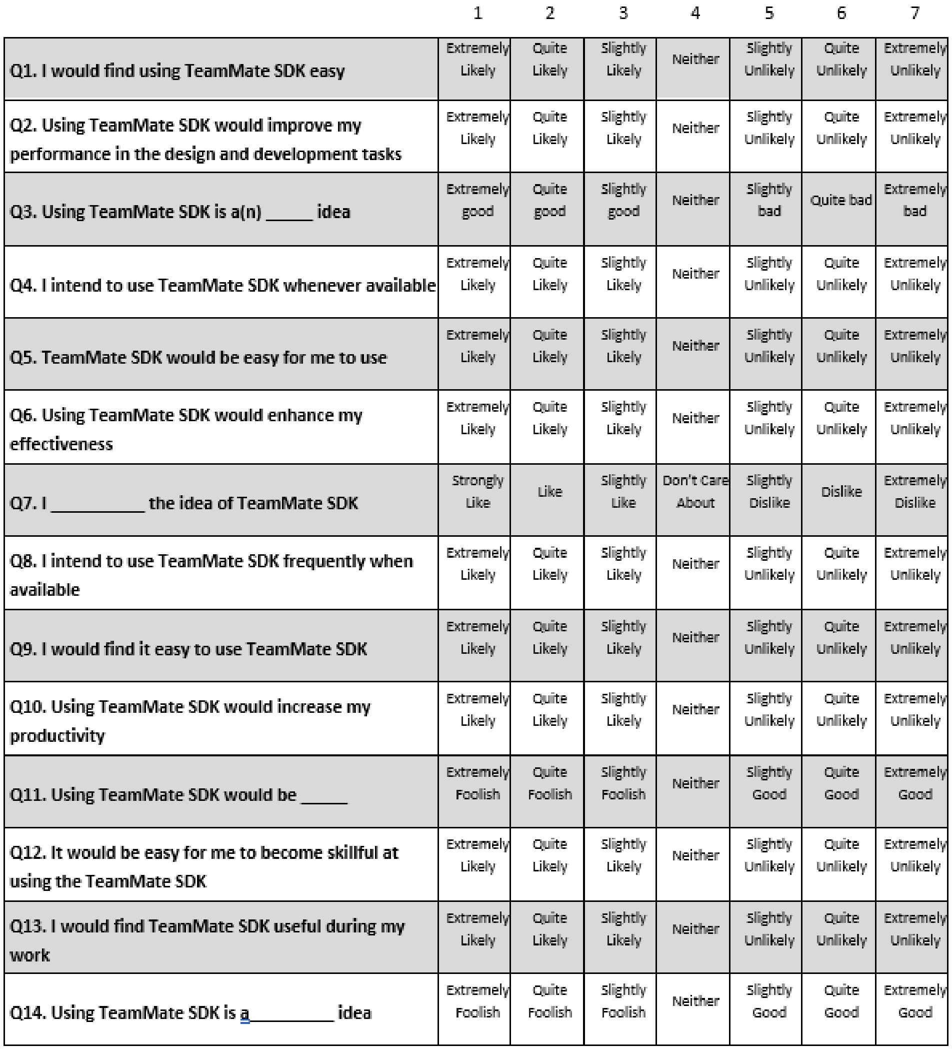
6.2.1 Participants

5 developers have been involved for each test site (for REL and HMT).

The target group has been selecting by considering developers with at least 3 years of experience in developing HW, FW and mobile applications.

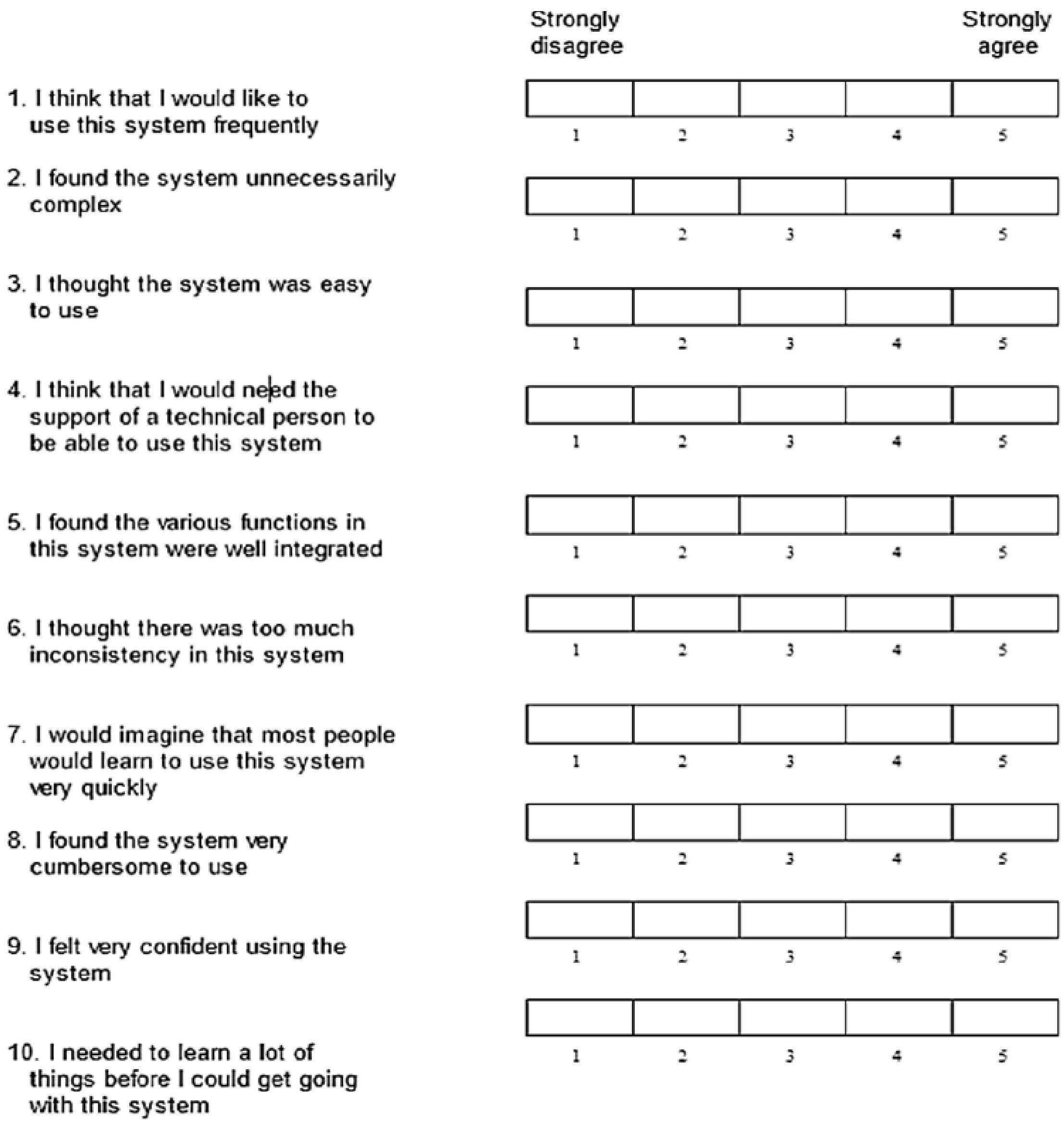
6.2.2 Questionnaires

System Usability Scale (SUS) and a Technology Acceptance Model (TAM) questionnaires have been used to measure respectively the usability and acceptability of the SDKs.



	1	2	3	4	5	6	7
Q1. I would find using TeamMate SDK easy	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q2. Using TeamMate SDK would improve my performance in the design and development tasks	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q3. Using TeamMate SDK is a(n) _____ idea	Extremely good	Quite good	Slightly good	Neither	Slightly bad	Quite bad	Extremely bad
Q4. I intend to use TeamMate SDK whenever available	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q5. TeamMate SDK would be easy for me to use	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q6. Using TeamMate SDK would enhance my effectiveness	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q7. I __________ the idea of TeamMate SDK	Strongly Like	Like	Slightly Like	Don't Care About	Slightly Dislike	Dislike	Extremely Dislike
Q8. I intend to use TeamMate SDK frequently when available	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q9. I would find it easy to use TeamMate SDK	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q10. Using TeamMate SDK would increase my productivity	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q11. Using TeamMate SDK would be _____	Extremely Foolish	Quite Foolish	Slightly Foolish	Neither	Slightly Good	Quite Good	Extremely Good
Q12. It would be easy for me to become skillful at using the TeamMate SDK	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q13. I would find TeamMate SDK useful during my work	Extremely Likely	Quite Likely	Slightly Likely	Neither	Slightly Unlikely	Quite Unlikely	Extremely Unlikely
Q14. Using TeamMate SDK is a_________ idea	Extremely Foolish	Quite Foolish	Slightly Foolish	Neither	Slightly Good	Quite Good	Extremely Good

Figure 13: System Usability Scale (SUS)



	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	□ 1	□ 2	□ 3	□ 4	□ 5
2. I found the system unnecessarily complex	□ 1	□ 2	□ 3	□ 4	□ 5
3. I thought the system was easy to use	□ 1	□ 2	□ 3	□ 4	□ 5
4. I think that I would need the support of a technical person to be able to use this system	□ 1	□ 2	□ 3	□ 4	□ 5
5. I found the various functions in this system were well integrated	□ 1	□ 2	□ 3	□ 4	□ 5
6. I thought there was too much inconsistency in this system	□ 1	□ 2	□ 3	□ 4	□ 5
7. I would imagine that most people would learn to use this system very quickly	□ 1	□ 2	□ 3	□ 4	□ 5
8. I found the system very cumbersome to use	□ 1	□ 2	□ 3	□ 4	□ 5
9. I felt very confident using the system	□ 1	□ 2	□ 3	□ 4	□ 5
10. I needed to learn a lot of things before I could get going with this system	□ 1	□ 2	□ 3	□ 4	□ 5

Figure 14: Technology Acceptance Model (TAM)

6.3 Results for TeamMate Component Extension SDK

We asked developers from other relevant projects (2 from Auto accept, and 3 from Intellimar projects, in total 3 men and 2 woman) to participate in the evaluation of TeamMate extension SDK. First the background information was provided to the participants, such as a brief explanation about the Automate project and the TeamMate system and the component-based structure of the developed TeamMate system in Automate project. After introduction of the concepts, TeamMate extension SDK was introduced, and the SDK with its manual were provided to the participants. Next, the participants were asked to work on a task prepared for the evaluation.

Task description:

To evaluate the TeamMate extension SDK, HMT prepared a simple component set with two components. One of the components was playing the role of a simulator, by providing the ego vehicle and alter vehicle information's on the network. This component was implemented by HMT and provided to the participants.

Second component, planned to play the role of a TeamMate component by a simple functionality, which was to read data from the network, calculates time to collision to the next approaching vehicle and sends the notification of dangerous situation as the output of the component. The development of this second component was the task for the participants.

The participants had to choose the input and output data structure and to build a component handler by defining the connection and serialization type and the configuration specifications.

Evaluation Results:

We used TAM and SUS questionnaire to evaluate TeamMate Extension SDK as explained in section 6.2.2. To fill the questionnaire the participants had to enter scores (1-7) with 7 being the highest option. E.g., under the column "Extremely likely" a "7" should be entered; under the "Extremely unlikely" a "1" should be entered.

Using TAM questionnaire, we studied criteria of ease of use, usefulness, attitude towards use and intention of use regarding technology acceptance model. Figure 15 depicts the results of the evaluation. The Intention to use of the participants were slightly likely, as the projects they were dealing with were not binding them to use a component based and communication-based approach. As shown in this figure, participants were in averages likely to use this SDK. The participants found the SDK quite useful (usefulness with the average score of Quite Likely), however the ease of use has more tendency to slightly likely (5) as to quite likely (6). According to the result of the evaluation and based on the feedback, we worked on improving the SDK with the goal of improving ease of use criteria.

For the SUS questionnaire, we obtained the average score of 74,4. According to this questionnaire if the score is higher than 68, the developed technology is considered as acceptable. Therefore, using the score of 74,4 implicate that the TeamMate Component Extension SDK was satisfactory for users.

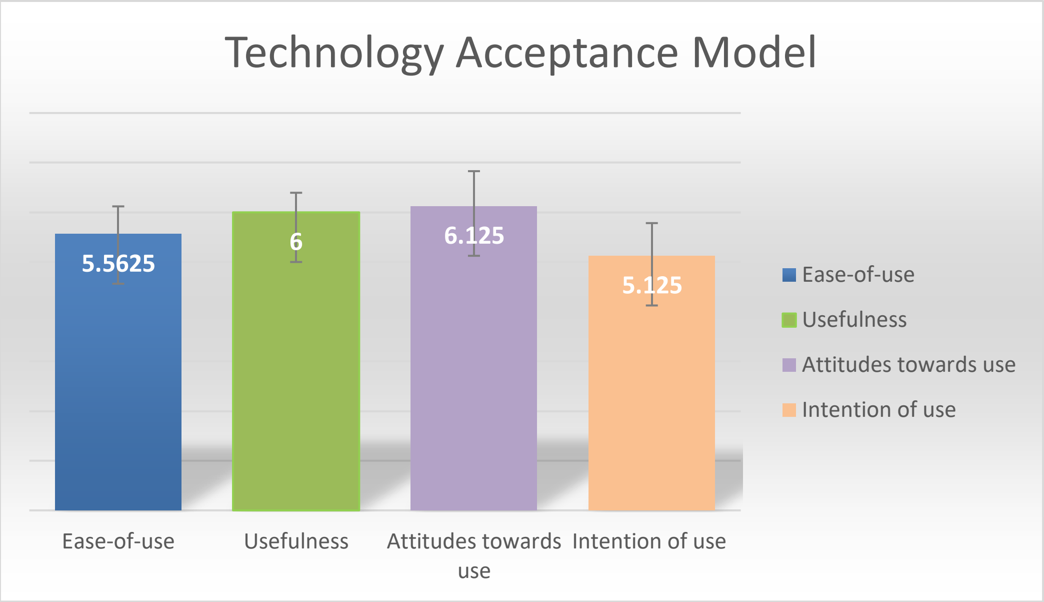


Figure 15: Technology acceptance model (TAM) for TeamMate Component Extension SDK.

The feedbacks of the participants and the actions to increase ease of use is explained in the following.

Feedbacks:

We gathered feedbacks from participants to further complete the "TeamMate component extension SDK" and thereby to improve Technology acceptance of users.

One useful feedback from participants was to include at least one example for each of the options offered by SDK. For example, to prepare sample components for each of the connection types and/or the serialization types. According to this feedback, HMT included several template components compounds and designed these templates such that they cover all the important aspects and configuration options of the TeamMate extension SDK.

Here are further suggestions we received from participants:

- I think you should add a short manual for step by step implementing new component to component framework
- If possible, it would be better to do less adjustment in the code, but more in a configuration file.
- Nice to have GUI or a simpler way to create components without changing and compiling the code base
- Nice to have Asynchronous "execute()" in components

We modified the manual and the source code to resolve the issues discovered during the discussion session with the participant at the end of the evaluation session.

6.4 Results for Mobile APP Extension SDK

The developers have been asked to use the SDKs (as well as the manual) to develop the same app developed in the project (i.e. Distributed HMI app).

The developers have been provided with the layouts and the state machine to clarify the behaviour of the app.

At the end of the development, the developers have been asked to answer the SUS and the TAM to measure the usability of the SDKs as well as its acceptance. Their comments have been also collected to provide useful feedback for future improvements.

The overall results of the tests conducted by REL will be part of the D6.4 Research Data.

The developers had a quite good experience using the SDKs and even though not all of them found them obvious and intuitive at first, they stated they could learn how to use them in some days of training and use. The TAM score has been calculated on a 7-point Likert scale with a 1-7 score range. The results of the TAM questionnaire show that the SDKs are well accepted by the developers. In particular, they were considered as useful (5,55), and the users were willing to adopt them (Intention to use = 5,3).

Despite the small user sample, this is highly significant since it highlights that the developers consider the option of actually using the tools when available.

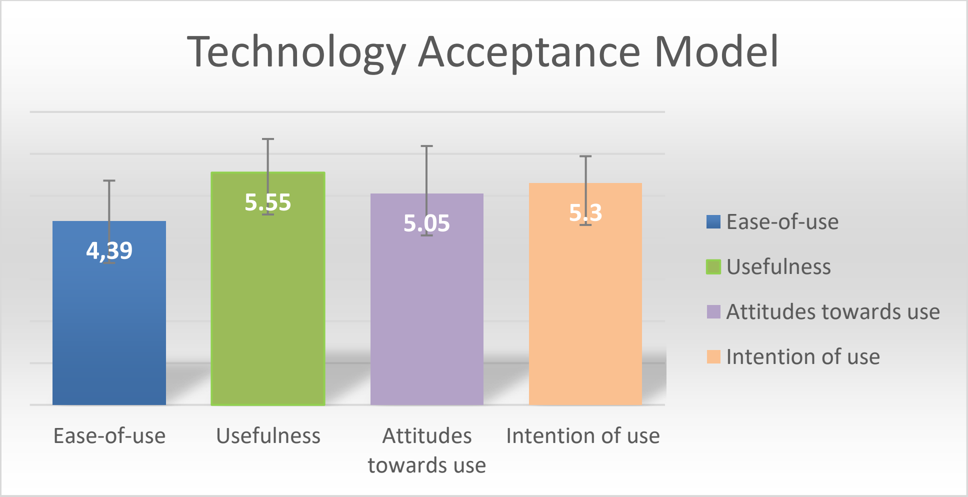


Figure 16: Technology acceptance model for Mobile APP Extension SDK

Also the SUS score (75) can be considered as satisfactory, since the threshold for score acceptability is widely considered as 68. In particular, some of the users felt confident in using the SDK, even if they believe it could take some time to learn how to effectively use them.

The detailed results of this test will be provided as part of the deliverable D6.4 (“Research Data”).

Some comments of the developers concerning the application of the SDKs:

1) the sequence of actions to be taken was not very clear to integrate the Gino module
2) a clear background of the HW and FW modules (e.g. BLE) was necessary to use the SDKs
3) A developer found the SDKs simple and clear, despite some initial difficulty in extending the existing GinoManager to create a new Manager (i.e. TeamMateManager).

7 Conclusions

In this deliverable we explained two complementary TeamMate SDKs developed by Automate partners, which enables third parties to extend the TeamMate system, by replacing or adding new functionalities.

The "TeamMate Component Extension SDK" developed by HMT, supports the extension of intrinsic functionalities, for example by replacing an enabler with its newest version or incorporating a completely new enabler to the system, which improve the overall decision making of the TeamMate system.

The “ThirdParty HMI/APP Extension SDK”, provides the extension of TeamMate system by external functionalities, for example, by building a new App upon the data obtained by TeamMate system or to build a new HMI system by accessing the output interface of the TeamMate system.

The SDKs has been used in practice to extend the TeamMate systems of the Automate partners. An example for deploying “TeamMate component extension SDK” is represented by the extension of REL TeamMate system by intention recognition component. As the second example, the ULM TeamMate system is extended by the online learning component using tools provided by “TeamMate component extension SDK”.

Since the objective of the test was to have a first impression of the SKDs from the developers, they were not properly trained and thus had some issues in using the SDKs effectively. Based on the experience of the researchers in using similar tools, a proper training is needed to effectively use them.

In general, the objective of the test can be considered met.

Appendix 1: User Manual and Source Code for TeamMate Extension SDK

The TeamMate SDK is an open-source software. The source code for the SDK includes multiple test suites and templates, which could be used to develop desired component compounds.

The source code for TeamMate Extension SDK and the prerequisite installation files can be downloaded from the following link:

https://my.hidrive.com/share/muak29qg40

The link includes a user manual in pdf, which explains how to build the Software in the IDE of choice. The manual provide instruction on the tools provided by the SDK and on design of components and the communication between components.

Appendix 2: User Manual for ThirdParty APP/HMI Extension SDK

The following classes are part of the ThirdParty Extension SDK:

class	DQData
class	DQError
class	DQObject
class	DQProperty
class	Gino
class	GinoContract
class	GinoManager
interface	GinoManagerListener

DQData

Public Member Functions

double	getDoubleValue ()

String	**getStringValue** ()
boolean	**getBoolValue** ()
long	**getTimestamp** ()
	DQData (**DQData** data)
	DQData (double value)
	DQData (String value)
	DQData (boolean value)
	DQData (byte[] value)

byte []	getRawValue ()

String	**toString** ()

String	**toHexString** ()

boolean	equals (Object o)

Constructor & Destructor Documentation

◆ DQData() [1/5]

com.dquid.sdk.core.DQData.DQData (DQData *data*)

Constructor to create a DQData from another DQData Object (Timestemp included). All other representations (String, boolean and raw) will be inferred automatically

Parameters

data The DQData to copy

◆ DQData() [2/5]

com.dquid.sdk.core.DQData.DQData (double *value*)

Constructor to put a double into **DQData**. All other representations (String, boolean and raw) will be inferred automatically

Parameters

value The double value

◆ DQData() [3/5]

com.dquid.sdk.core.DQData.DQData (String *value*)

Constructor to put a String into **DQData**. All other representations (double, boolean and raw) will be inferred automatically

Parameters

value The String value

◆ DQData() [4/5]

com.dquid.sdk.core.DQData.DQData (boolean *value*)

Constructor to put a boolean into **DQData**. All other representations (double, String and raw) will be inferred automatically

Parameters

value The boolean value

◆ DQData() [5/5]

com.dquid.sdk.core.DQData.DQData (byte [] *value*)

Constructor to put a byte[] into DQData. All other representations (double, String and boolean) will be inferred automatically The byte[] is cloned.

Parameters

value The byte[] value

Member Function Documentation

◆ equals()

boolean com.dquid.sdk.core.DQData.equals (Object *o*)

Return true if the DQData values are the same as the passed DQData object

Parameters

o the DQData to compare to.

Returns

true if the two objects havethe same values, false otherwise

◆ getBoolValue()

boolean com.dquid.sdk.core.DQData.getBoolValue ( )

Returns

the boolValue

◆ getDoubleValue()

double com.dquid.sdk.core.DQData.getDoubleValue ( )

Returns

the doubleValue

◆ getRawValue()

byte [] com.dquid.sdk.core.DQData.getRawValue ( )

Returns

the rawValue

◆ getStringValue()

String com.dquid.sdk.core.DQData.getStringValue ( )

Returns

the stringValue

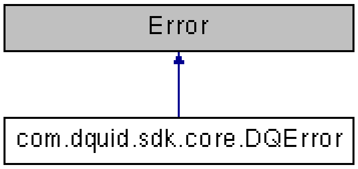
◆ getTimestamp()

long com.dquid.sdk.core.DQData.getTimestamp ( )

Returns

the timestamp

DQError



Public Member Functions

	DQError (int errCode)
	DQError (String info)
	DQError (int errCode, String info)
boolean	**isBondingError** ()

Public Attributes

final int	**code**

final String	**description**

Static Public Attributes

static final int	**E_GENERIC** = Integer.MAX_VALUE
static final int	**E_TIMEOUT** = 1
static final int	**E_UNSUPPORTED** = 2
static final int	**E_NOTENABLED** = 3
static final int	**E_BOND_ERROR** = 4

DQObject

Public Member Functions

String	getCreatedBy ()

DQObjectPicture	**getPicture** ()
String	**getVisibility** ()
String	**getName** ()
double	**getLat** ()
double	**getLon** ()
String	**getObjectId** ()
HashMap< String, **DQProperty** >	**getPropertiesByName** ()
void	**setListener** (DQObjectListener listener)

synchronized boolean	**connect** ()
synchronized boolean	**disconnect** ()
boolean	**isConnected** ()
boolean	**read** (String propertyName, boolean requiresAck)
boolean	**readProperties** (Collection< String > propertiesNames, boolean requiresAck)
boolean	**subscribe** (String propertyName, boolean requiresAck)
boolean	**subscribeToProperties** (Collection< String > propertiesNames, boolean requiresAck)

boolean	**subscribeToProperties** (Collection< String > propertiesNames, int rate, int sampleMode, int saveMode)
boolean	**unsubscribe** (String propertyName, boolean requiresAck)
boolean	**unsubscribeFromProperties** (Collection< String > propertiesNames, boolean requiresAck)
boolean	**fastSubscribeToProperties** (Collection< String > propertiesNames, int rate, boolean requiresAck)
boolean	**fastUnsubscribeFromProperties** (Collection< String > propertiesNames, boolean requiresAck)
boolean	**write** (**DQData** dqdata, String propertyName, boolean requiresAck)

boolean	**writeToProperties** (Map< String, **DQData** > propertiesAndDatas, boolean requiresAck)
boolean	**write** (**DQData** dqdata, String propertyName, int mode, int rate)
boolean	**equals** (Object o)

Detailed Description

This class represents a DQuid Object (any physical object tagged using the DQuid technology). It shows all the needed object information and contains a dictionary of all the object properties.

The class also provides methods to (dis)connect and read/write the properties of the object.

It has a listener object (that need to implement the 'DQObjectListener' interface) that allows to handle events regarding this object.

Member Function Documentation

◆ connect()

synchronized boolean com.dquid.sdk.core.DQObject.connect ( ) inline

Tries to connect to the object. 'onErrorOccurred' method of DQObjectListener may be called consequently to any potential issue.

Returns

true if the connection request was issued correctly, false otherwise;

◆ disconnect()

synchronized boolean com.dquid.sdk.core.DQObject.disconnect ( )

Tries to connect to the object. 'onErrorOccurred' method of DQObjectListener may be called consequently to any potential issue.

Returns

true if the disconnection request was issued correctly, false otherwise;

◆ getCreatedBy()

String com.dquid.sdk.core.DQObject.getCreatedBy ( ) inline

Returns

the createdBy

◆ getLat()

double com.dquid.sdk.core.DQObject.getLat ( ) inline

Returns

the lat

◆ getLon()

double com.dquid.sdk.core.DQObject.getLon ( ) inline

Returns

the lon

◆ getName()

String com.dquid.sdk.core.DQObject.getName ( ) inline

Returns

the name

◆ getObjectId()

String com.dquid.sdk.core.DQObject.getObjectId ( ) inline

Returns

the objectId

◆ getPicture()

DQObjectPicture com.dquid.sdk.core.DQObject.getPicture ( ) inline

Returns

the picture

◆ getPropertiesByName()

HashMap<String, **DQProperty**> com.dquid.sdk.core.DQObject.getPropertiesByName ( ) inline

Returns

the propertiesByName

◆ getVisibility()

```
String com.dquid.sdk.core.DQObject.getVisibility ( )                inline
```

Returns

the visibility

◆ isConnected()

```
boolean com.dquid.sdk.core.DQObject.isConnected ( )                 inline
```

Returns

true if the sdk is connected to that DQuid Object, false otherwise

◆ read()

```
boolean com.dquid.sdk.core.DQObject.read ( String   propertyName,
                                           boolean  requiresAck
                                         )                          inline
```

Sends a read request for a certain property of the object

The read value is sent back to the listener object using the 'onPropertyReceivedData' method . 'onErrorOccurred' may be called consequently to any potential issue.

Parameters

propertyName The name of the property whose value we are requesting

Returns

True if the request was completed, false otherwise

◆ readProperties()

boolean com.dquid.sdk.core.DQObject.readProperties (Collection< String > *propertiesNames*,
boolean *requiresAck*
) inline

Sends a read request for multiple properties of the object

The read values are sent back to the listener object using the 'onPropertyReceivedData' method . 'onErrorOccurred' may be called consequently to any potential issue.

Parameters

propertiesNames A list of properties names

Returns

True if the request was completed for at least some properties belonging to one DQUnit, false otherwise.

◆ setListener()

void com.dquid.sdk.core.DQObject.setListener (DQObjectListener *listener*) inline

Parameters

listener the listener to set

◆ subscribe()

boolean com.dquid.sdk.core.DQObject.subscribe	(	String	*propertyName,*
		boolean	*requiresAck*
	)		inline

Sends a subscribe request for a certain property of the object

After subscription the method 'onPropertyReceivedData' of the listener object will be called every time the subscribed property changes. 'onErrorOccurred' may be called consequently to any potential issue.

Parameters

propertyName The name of the property whose value we are requesting

Returns

True if the request was completed, false otherwise

◆ subscribeToProperties() [1/2]

boolean com.dquid.sdk.core.DQObject.subscribeToProperties	(	Collection< String >	*propertiesNames,*	
		boolean	*requiresAck*	
	)			inline

Sends a subscribe request for multiple properties of the object

The subscribed values are sent back to the listener object using the 'onPropertyReceivedData' method . 'onErrorOccurred' may be called consequently to any potential issue.

Parameters

propertiesNames A list of properties names

Returns

True if the request was completed for at least some properties belonging to one DQUnit, false otherwise.

◆ subscribeToProperties() [2/2]

boolean com.dquid.sdk.core.DQObject.subscribeToProperties	(	Collection< String >	*propertiesNames,*	
		int	*rate,*	
		int	*sampleMode,*	
		int	*saveMode*	inline
	)			

Start to receive updates for the selected CAN properties

The subscribed values are sent back to the listener object using the 'onPropertyReceivedData' method . 'onErrorOccurred' may be called consequently to any potential issue.

Parameters

properties The name of the properties (CAN signal) whose value we are requesting

rate The speed at which request updates

sampleMode The sample mode to use for parsing data

saveMode The save mode for saving into flash the received CAN data

Returns

false in case of error, true otherwise

◆ unsubscribe()

```
boolean com.dquid.sdk.core.DQObject.unsubscribe ( String   propertyName,
                                                  boolean  requiresAck
                                                )                          inline
```

Sends an unsubscribe request for a certain property of the object

After unsubscription the listener will no more be notified about property cahnges. 'onErrorOccurred' may be called consequently to any potential issue.

Parameters

propertyName The name of the property whose value we are requesting

Returns

True if the request was completed, false otherwise

◆ unsubscribeFromProperties()

```
boolean
com.dquid.sdk.core.DQObject.unsubscribeFrom   Collection<    propertiesNa
Properties                                   ( String >       mes,

                                               boolean        requiresAck
                                                                            inlin
                                             )                              e
```

Sends an unsubscribe request for multiple properties of the object

The listener will not be notified anymore about properties change after this call. 'onErrorOccurred' may be called consequently to any potential issue.

Parameters

propertiesNames A list of properties names

Returns

True if the request was completed for at least some properties belonging to one DQUnit, false otherwise.

◆ write() [1/2]

```
boolean com.dquid.sdk.core.DQObject.write ( DQData   dqdata,

                                            String   propertyName,

                                            boolean  requiresAck

                                          )                              inline
```

Sends a write request (data type is automatically inferred) for a certain property of the object.

'onErrorOccurred' callback may be called consequently to any potential issue.

Parameters

dqdata	The generic object to be written
propertyName	The name of the property whose value we are trying to write

Returns

True if the request was completed, false otherwise

◆ write() [2/2]

```
boolean com.dquid.sdk.core.DQObject.write ( DQData  dqdata,
                                            String  propertyName,
                                            int     mode,
                                            int     rate
                                          )                         inline
```

Sends a write request for a certain CAN property of the object

'onErrorOccurred' callback may be called subsequentially to any potential issue.

Parameters

dqdata	The **DQData** to write

propertyName The name of the DQProperty to write to

rate The sample time at which wite the data

Returns

YES if athe request was forwarded

◆ writeToProperties()

boolean com.dquid.sdk.core.DQObject.writeToProperties	(	Map< String, DQData >	*propertiesAndDatas,*	inline
		boolean	*requiresAck*	
	)			

Sends a write request (data type is automatically inferred) for multiple properties of the object.

'onErrorOccurred' callback may be called consequently to any potential issue.

Parameters

propertiesAndDatas A map with all properties and all datas to be written

Returns

True if the request was completed, false otherwise

DQProperty

Public Member Functions

String	getName ()
String	getUm ()
DQDataType	getReadDataType ()
DQDataType	getWriteDataType ()
boolean	isReadable ()
boolean	isWritable ()
boolean	isAvailable ()

boolean	equals (Object o)

Protected Attributes

DQUnit	**dquidUnit**

String	um

DQDataType	readDataType

DQDataType	writeDataType

boolean	readable

boolean	writable

Detailed Description

This class represents a property of a DQuid Object. It contains the property name, unit of measure, and the information about readability/writeability. It also shows which data type has to be used when reading/writing to it.

Member Function Documentation

◆ equals()

```
boolean com.dquid.sdk.core.DQProperty.equals ( Object o )
```

inline

Return true if the DQProperty is the same as the passed DQProperty object

Parameters

o the DQProperty to compare to.

Returns

true if the two objects are the same, false otherwise

◆ getName()

```
String com.dquid.sdk.core.DQProperty.getName ( )
```

inline

Returns

the name

◆ getReadDataType()

```
DQDataType com.dquid.sdk.core.DQProperty.getReadDataType ( )
```

inline

Returns

the readDataType

◆ getUm()

String com.dquid.sdk.core.DQProperty.getUm ( ) inline

Returns

the um

◆ getWriteDataType()

DQDataType com.dquid.sdk.core.DQProperty.getWriteDataType ( ) inline

Returns

the writeDataType

◆ isAvailable()

boolean com.dquid.sdk.core.DQProperty.isAvailable ( ) inline

Returns

true if the property is available, false otherwise

◆ isReadable()

boolean com.dquid.sdk.core.DQProperty.isReadable ( ) inline

Returns

the readable

◆ isWritable()

boolean com.dquid.sdk.core.DQProperty.isWritable ( ) inline

Returns

the writable

Member Data Documentation

◆ readable

boolean com.dquid.sdk.core.DQProperty.readable protected

True if the data is readable

◆ readDataType

DQDataType com.dquid.sdk.core.DQProperty.readDataType protected

The readable data type (a **DQData** can be read using a data type and written using another)

◆ um

String com.dquid.sdk.core.DQProperty.um protected

The unit of measure of the data provided by this property

◆ writable

boolean com.dquid.sdk.core.DQProperty.writable protected

True if the data is writable

◆ writeDataType

DQDataType com.dquid.sdk.core.DQProperty.writeDataType protected

The writable data type

Gino

Public Member Functions

String	**getName** ()
String	**getSerial** ()
HashMap< String, DQProperty >	**getPropertiesByName** ()
HashMap< String, DQProperty >	getPropertiesForChannel (int channelType)
boolean	**isConnected** ()
boolean	**isReady** ()

Static Public Member Functions

static String	**nameForCharacteristicUuid** (String uuid)
static boolean	**hasStandardPropertiesUpdate** (Map< String, List< [DQData](#) >> propertiesUpdate)

Static Public Attributes

static final String	**CAN0_DBC_FILENAME_KEY** = "can0_dbc_filename"
static final String	**CAN1_DBC_FILENAME_KEY** = "can1_dbc_filename"

Member Function Documentation

◆ getPropertiesForChannel()

HashMap<String,DQProperty> com.dquid.sdk.core.Gino.getPropertiesForChannel (int *channelType*) inline

Returns

the properties for specific channelType

GinoContract

Classes

enum	**Advertisement**
interface	**CompanyIdentifier**
interface	**DeveloperKeyStore**
enum	**GpsFixQuality**
enum	**GpsFixType**
interface	**PropertyDef**
interface	**ProtocolFunction**

enum	**SampleMode**
enum	**SaveMode**
enum	**StandardProperties**
enum	**StandardProtocolFunction**
enum	**SupportedCompany**

Static Public Attributes

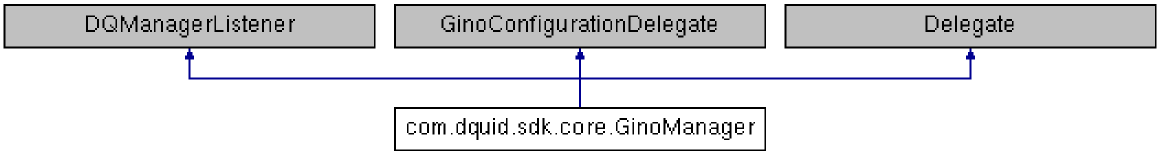
static final String	**PROPERTY_SUBSCRIBE** = "subscribeProperty"
static final String	**PROPERTY_WRITE** = "writeProperty"

static final String	**PROPERTY_READWRITE** = "readWriteProperty"
static final String []	**STANDARD_PROPERTIES** = new String[]{PROPERTY_SUBSCRIBE, PROPERTY_WRITE, PROPERTY_READWRITE}
static final int	**STANDARD_CHANNEL_TYPE** = 255
static final int	**CAN0_CHANNEL_TYPE** = 0
static final int	**CAN1_CHANNEL_TYPE** = 1
static String	**PROTO_IN_CHAR** = "85279D3F-55E7-4963-8F34-09C40F0D935A"
static String	**PROTO_OUT_CHAR** = "85279D3F-55E7-4963-8F34-09C40F0D935B"

static String	**PROTO_OUT_CHAR2** = "85279D3F-55E7-4963-8F34-09C40F0D935C"

static String	**PROTO_SERVICE** = "88FE7F19-B739-4029-909B-38A74465DD7B"

GinoManager



Public Member Functions

GinoManagerListener	**getGinoManagerListener** ()

Set< GinoContract.CompanyIdentifier >	**getScanIncludedCompanies** ()

void	**addGinoManagerListener** (**GinoManagerListener** listener)
void	**removeGinoManagerListener** (**GinoManagerListener** listener)
void	**startDiscovery** ()
void	**stopDiscovery** ()
void	**onNewObjectDiscovered** (**DQObject** object)
void	**onObjectConnected** (**DQObject** object)
void	**onObjectConnectionFail** (**DQObject** object, **DQError** error)
void	**onObjectPropertiesUpdated** (**DQObject** object)

void	**onObjectDisconnected** (**DQObject** object)
void	**onErrorOccurred** (**DQError** error)
void	**onDataReceived** (**DQObject** object, Map< **DQProperty**, List< **DQData** >> update)
void	**onDataReceived** (**DQObject** object, List< Pair< **DQProperty**, **DQData** >> updateSequence)
void	**onDataReceived** (**DQObject** object, int size, Date date)
void	**onWriteRequestACKed** (**DQObject** object, Map<? extends **DQProperty**, ? extends **DQData** > propertiesAndDataAcked)

void	**onCommandRequestACKed** (DQObject object, DQRequestType requestType, Collection<? extends DQProperty > propertiesAcked)
void	**onWriteRequestACKError** (DQObject object, Map<? extends DQProperty, ? extends DQData > propertiesAndDataNotAcked, DQError error)
void	**onCommandRequestACKError** (DQObject object, DQRequestType requestType, Collection<? extends DQProperty > propertiesNotAcked, DQError error)
Context	**provideApplicationContext** ()
void	**onDiscoveryError** (DQError e)
boolean	**connectToGino** (Gino newGino)

boolean	**disconnectFrom** (Gino gino)
boolean	**readProperty** ()
boolean	**readProperty** (String property)
boolean	**subscribeToProperty** ()
boolean	**subscribeToProperty** (String property, int rate, int sampleMode, int saveMode)
boolean	**subscribeToProperties** (Collection< String > properties, int rate, int sampleMode, int saveMode)
boolean	**unsubscribeToProperty** ()
boolean	**unsubscribeFromProperty** (String property)

boolean	**unsubscribeFromProperties** (List< String > properties)
boolean	**writeProperty** (String property, byte[] data)
boolean	**writeProperty** (GinoContract.PropertyDef propertyDef, byte[] data)
boolean	**writeProperty** (String property, byte[] data, DQWriteMode mode, int rate)
boolean	**writeProperty** (GinoContract.PropertyDef propertyDef, byte[] data, DQWriteMode mode, int rate)
boolean	**requestBootloaderVersion** ()
boolean	**requestApplicationLibraryVersion** ()

Static Public Member Functions

static GinoManager	**getSharedInstance** (Context ctx, GinoManagerListener ginoManagerListener, String developerKey)
static GinoManager	**getSharedInstance** (Context ctx, GinoManagerListener ginoManagerListener, GinoPlugin ginoPlugin, String developerKey)
static GinoManager	**getSharedInstance** (Context ctx, GinoManagerListener ginoManagerListener, GinoPlugin ginoPlugin, String developerKey, int targetSerialNumber)

Protected Member Functions

	GinoManager (GinoManagerListener ginoManagerListener)
void	**init** (Context ctx, GinoPlugin customConfiguration, String developerKey, int targetSerialNumber)

void	**subscribeCommandHelper** (CommandOperationHelper commandHelper)

boolean	**subscribeToProperty** (GinoContract.PropertyDef propertyDef)

Member Function Documentation

◆ readProperty() [1/2]

boolean com.dquid.sdk.core.GinoManager.readProperty ( ) inline

Sends a read request for Gino.PROPERTY_READWRITE

The read value is sent back to the delegate object through the proper callback method. 'onErrorOccurred' callback may be called subsequentially to any potential issue.

◆ readProperty() [2/2]

boolean com.dquid.sdk.core.GinoManager.readProperty (String *property*) inline

Sends a read request for a certain property of the object

The read value is sent back to the delegate object through the proper callback method. 'onErrorOccurred' callback may be called subsequentially to any potential issue.

Parameters

property The name of the property whose value we are requesting

◆ requestApplicationLibraryVersion()

boolean com.dquid.sdk.core.GinoManager.requestApplicationLibraryVersion ( ) inline

Request Application and Library versions

Returns

YES if the request was forwarded

◆ requestBootloaderVersion()

boolean com.dquid.sdk.core.GinoManager.requestBootloaderVersion ( ) inline

Request BootloaderVersion

Returns

YES if the request was forwarded

◆ subscribeToProperties()

boolean com.dquid.sdk.core.GinoManager.subscribeToProperties	(	Collection< String >	*properties,*
		int	*rate,*
		int	*sampleMode,*
		int	*saveMode*
	)		inline

Start to receive updates for the selected properties

Parameters

properties	The names of the properties you want to subscribe
rate	The rate which you will receive the updates, in milliseconds

Returns

NO in case of error, YES otherwise Start to receive updates for the selected CAN properties

Parameters

properties	The names of the properties you want to subscribe
rate	The speed at which request updates
sampleMode	The sample mode to use for parsing data
saveMode	The save mode for saving into flash the received CAN data

Returns

NO in case of error, YES otherwise

◆ subscribeToProperty() [1/2]

boolean com.dquid.sdk.core.GinoManager.subscribeToProperty ( ) inline

Start to receive updates for Gino.PROPERTY_SUBSCRIBE

Returns

NO in case of error, YES otherwise

◆ subscribeToProperty() [2/2]

```
boolean
com.dquid.sdk.core.GinoManager.subscribeToProperty    ( String  property,
                                                        int     rate,
                                                        int     sampleMode,
                                                        int     saveMode
                                                      )                    inline
```

Start to receive updates for the selected CAN property

Parameters

property	The name of the property (CAN signal) whose value we are requesting
rate	The speed at which request updates
sampleMode	The sample mode to use for parsing data
saveMode	The save mode for saving into flash the received CAN data

Returns

NO in case of error, YES otherwise TODO:

subscribeToProperty:withRate:sampleMode:andSaveMode: MUST be implemented at driver level, abstract or perhaps with empty implementations in DriverModule and with actual implementation in DQuidV2DriverModule.

DQuidObject calls driver which creates a CanDataRequestMessage and sends it using state.sendDataMessage Start to receive updates for the selected CAN property

Parameters

property	The name of the property (CAN signal) whose value we are requesting
rate	The speed at which request updates
sampleMode	The sample mode to use for parsing data
saveMode	The save mode for saving into flash the received CAN data

Returns

NO in case of error, YES otherwise TODO: subscribeToProperty:withRate:sampleMode:andSaveMode: MUST be implemented at driver level, abstract or perhaps with empty implementations in DriverModule and with actual implementation in DQuidV2DriverModule.

DQuidObject calls driver which creates a CanDataRequestMessage and sends it using state.sendDataMessage Start to receive updates for the selected GPS property

Parameters

property	The name of the property (GPS signal) whose value we are requesting
rate	The speed at which request updates
sampleMode	The sample mode to use for parsing data
saveMode	The save mode for saving into flash the received CAN data

Returns

NO in case of error, YES otherwise

◆ unsubscribeFromProperties()

boolean com.dquid.sdk.core.GinoManager.unsubscribeFromProperties (List< String > *properties*) inline

Stop receiving updates from the selected properties

Parameters

properties The names of the properties you want to unsubscribe

Returns

NO in case of error, YES otherwise

◆ unsubscribeFromProperty()

boolean com.dquid.sdk.core.GinoManager.unsubscribeFromProperty (String *property*) inline

Stop receiving updates from the selected property

Parameters

property The names of the property you want to unsubscribe

Returns

NO in case of error, YES otherwise

◆ unsubscribeToProperty()

boolean com.dquid.sdk.core.GinoManager.unsubscribeToProperty ( ) inline

Stop receiving updates from Gino.PROPERTY_SUBSCRIBE

Returns

NO in case of error, YES otherwise

◆ writeProperty() [1/2]

boolean com.dquid.sdk.core.GinoManager.writeProperty (String *property*,
byte [] *data*
) inline

Start to receive updates for all the properties of the connected **Gino**

Parameters

rate The rate which you will receive the updates, in milliseconds

Returns

NO in case of error, YES otherwise Stop receiving updates from the **Gino**

NO in case of error, YES otherwise Sends a write request (data type is selected according to the writeDataType of the property) for a certain property of the object

'onErrorOccurred' callback may be called subsequentially to any potential issue.

Parameters

data The **DQData** to write

property The name of the **DQProperty** to write to

Returns

YES if athe request was forwarded

◆ writeProperty() [2/2]

```
boolean
com.dquid.sdk.core.GinoManager.writeProperty    ( String        property,
                                                  byte []       data,
                                                  DQWriteMode   mode,
                                                  int           rate
                                                )                        inline
```

Sends a write request for a certain CAN property of the object

'onErrorOccurred' callback may be called subsequentially to any potential issue.

Parameters

data The **DQData** to write

property The name of the **DQProperty** to write to

rate The sample time at which wite the data

Returns

YES if athe request was forwarded

GinoManagerListener

Public Member Functions

void	**onGinoManagerInit** (int initStatus)
void	**onGinoDiscovered** (**Gino** newGino)
void	**onConnectionEstablishedForGino** (**Gino** gino)
void	**onConnectionFailedForGino** (**Gino** gino, **DQError** error)
void	**onDisconnectionFromGino** (**Gino** gino)
void	**onGinoReceivedUpdates** (**Gino** gino, Map< String, List< **DQData** >> propertiesUpdate)
void	**onGinoReceivedUpdates** (**Gino** gino, List< Pair< String, **DQData** >> updateSequence)

void	**onGinoReceivedDataOfSize** (**Gino** gino, int size, Date date)
void	**onErrorOccurred** (**DQError** error)
void	**onGinoReceivedApplicationVersion** (**Gino** gino, Version appVersion, Version libVersion)
void	**onGinoReceivedBootloaderVersion** (**Gino** gino, Version bootloaderVersion)

Member Function Documentation

◆ onConnectionEstablishedForGino()

void com.dquid.sdk.core.GinoManagerListener.onConnectionEstablishedForGino (**Gino** *gino*)

Method called every time a connection to a **Gino** has been correctly established

Parameters

gino The connected **Gino**

◆ onConnectionFailedForGino()

void com.dquid.sdk.core.GinoManagerListener.onConnectionFailedForGino (**Gino** *gino*,
DQError *error*
)

Method called every time a connection to a **DQObject** fails

Parameters

gino The **Gino**

◆ onDisconnectionFromGino()

void com.dquid.sdk.core.GinoManagerListener.onDisconnectionFromGino (**Gino** *gino*)

Method called every time a disconnection to a **DQObject** happens.

Parameters

gino The disconnected **Gino**

◆ onErrorOccurred()

void com.dquid.sdk.core.GinoManagerListener.onErrorOccurred (**DQError** *error*)

Unspecified DQuid error

Parameters

error

◆ onGinoReceivedApplicationVersion()

void com.dquid.sdk.core.GinoManagerListener.onGinoReceivedApplicationVersion	(	Gino	*gino*,
		Version	*appVersion*,
		Version	*libVersion*
	)		

Mathod called when a response to GetApplicationVersion arrives

◆ onGinoReceivedBootloaderVersion()

void com.dquid.sdk.core.GinoManagerListener.onGinoReceivedBootloaderVersion	(	Gino	*gino*,
		Version	*bootloaderVersion*
	)		

Mathod called when a response to GetBootloaderVersion arrives

◆ onGinoReceivedDataOfSize()

void com.dquid.sdk.core.GinoManagerListener.onGinoReceivedDataOfSize	(**Gino**	*gino,*
	int	*size,*
	Date	*date*
	)	

Mathod called every time a new packet is received by the **Gino**

Parameters

gino The connected **Gino**

size The size of the received packet

date The NSDate at which the packet has been received

◆ onGinoReceivedUpdates() [1/2]

void com.dquid.sdk.core.GinoManagerListener.onGinoReceivedUpdates	(**Gino**	*gino,*
	Map< String, List< **DQData** >>	*propertiesUp date*
	)	

Method called every time a property has been updated by the **Gino**

Parameters

gino The Gino that receive the updates

propertiesUpdate A dictionary containing all the data received. The key is the name of the property updated, the object is a list of DQData

◆ onGinoReceivedUpdates() [2/2]

```
void
com.dquid.sdk.core.GinoManagerListener.onGinoReceivedUpdates ( Gino                          gino,
                                                               List< Pair< String, DQData >>  updateSequence
                                                             )
```

Method called every time properties got updated by the Gino

Parameters

gino The Gino that receive the updates

updateSequence A list containing the updates preserving arrival order.

Appendix 3: Source code for ThirdParty APP/HMI Extension SDK

```
//  AutomateManager.m

#import "AutomateManager.h"
#define GINO_SERIAL @"9F401FDE"

#define Instrument_ID @"CAN0.MabxToHmiRelab.Instrument_ID"
#define LeftLineConfidence @"CAN0.MabxToHmiRelab.LeftLineConfidence"
#define RightLineConfidence @"CAN0.MabxToHmiRelab.RightLineConfidence"
#define RoundBoundApproachingSt @"CAN0.MabxToHmiRelab.RoundBoundApproachingSt"
#define AttentionState @"CAN0.MabxToHmiRelab.AttentionState"

@import GinoFramework;
@import DQuidSdk;
@interface AutomateManager()<DQGinoManagerDelegate> {
    NSMutableDictionary *dictForAlarm;
    AutomateWarning_t prevWarning;
    double statusFSM_Automate_value_prev;
    bool isFirst_statusFSM_Automate_value_prev;
    NSTimeInterval tor_initial_time;
    bool tor_condition_check;
}

@end

@implementation AutomateManager
+ (AutomateManager *)sharedController{
    static dispatch_once_t pred;
    static AutomateManager *shared = nil;

    dispatch_once(&pred, ^{
        shared = [[AutomateManager alloc] init];
    });
    return shared;
}


- (instancetype)init {

    self = [super init];
    if (self) {
        dictForAlarm = [[NSMutableDictionary alloc] init];

        [DQGINOMANAGER addDelegate:self];
        [DQGINOMANAGER
setDeveloperKey:@"CyiLDpSIsTPivD8DfWaxwBHmwBonE6yUjDG3P6o1ChjPYxhT4KMZXiwz/X
tp1d0xIHY0fy2560PFQri3heUqHQ=="];
        [DQGINOMANAGER setShoudAvoidPropertyNameDuplicates:YES];

        [DQGINOMANAGER importDbc:[[NSBundle mainBundle]
pathForResource:@"HmiRelabPrivate" ofType:@"dbc"] onChannel:0];
        [self startSearchingGino];

        tor_condition_check = false;

        //Initialization
        prevWarning = AutomateWarning_NONE;
    }
    return self;
}


- (void)startSearchingGino {
    [DQGINOMANAGER startSearchingGino];
}

#pragma mark DQGinoManagerDelegate
-(void)onNewGinoDiscovered:(Gino *)gino {
   // if ([gino.serial isEqualToString:GINO_SERIAL]) {
        [DQGINOMANAGER stopSearchingGino];
        [DQGINOMANAGER connectToGino:gino];
//    }
//    else
//        [_delegate onDifferentGinoDiscovered];
}

- (void) onConnectionEstablishedForGino:(Gino *)gino {
    NSArray *list = [[NSArray alloc] initWithObjects:Instrument_ID,
                    RoundBoundApproachingSt,
                    LeftLineConfidence,
                    RightLineConfidence,
                    AttentionState,
                    nil];
    [DQGINOMANAGER subscribeToProperties:list withRate:500 sampleMode:0
andSaveMode:0];
```

```
    [_delegate onConnectedToGino];

}

- (void) onConnectionFailedForGino:(Gino *)gino {
    [_delegate onDisconnectedFromGino];

}


- (void) onDisconnectionFromGino:(Gino *)gino {
    [_delegate onDisconnectedFromGino];

}


- (void) onGino:(Gino *)gino receivedUpdates:(NSDictionary*)updates {

    for (id propertyName in [updates allKeys]) {

        DQData *data = [[updates objectForKey:propertyName] objectAtIndex:0];
        //NSLog(@"Received %@  -value: %f",propertyName, data.doubleValue  );

        [dictForAlarm setObject:data forKey:propertyName];

    }

    [self checkAlarm:dictForAlarm];


}


- (void)checkAlarm:(NSDictionary*)updates {

    AutomateWarning_t curWarning = AutomateWarning_NONE;

    //Check if data is available, otherwise skip
    DQData* instrument_ID_data = [updates objectForKey:Instrument_ID];
    DQData* leftLineConfidence_data = [updates objectForKey:LeftLineConfidence];
    DQData* rightLineConfidence_data = [updates objectForKey:RightLineConfidence];
    DQData* roundBoundApproachingSt_data = [updates objectForKey:
RoundBoundApproachingSt];
    DQData* attentionState_data = [updates objectForKey: AttentionState];
    if (instrument_ID_data == nil ||
        leftLineConfidence_data == nil ||
        rightLineConfidence_data == nil ||
        roundBoundApproachingSt_data == nil ||
        attentionState_data == nil)
        return;

/*
 o For Take Lateral Control
 - RoundBoundApproachingSt == 1 AND
 - Istrument_ID !=2 AND AttentionState ==4
 o For Take Over Request
 - (LeftLineConfidence == 0) AND (RightLineConfidence == 0)
          for more than 4 secs
 - Istrument_ID !=2 AND AttentionState ==4

 */

    if (instrument_ID_data.doubleValue !=2 && attentionState_data.doubleValue == 4) {
        //TLCR warning
        if (roundBoundApproachingSt_data.doubleValue == 1) {
            curWarning = AutomateWarning_TLCR;
            tor_condition_check = false;
        }
        else {
        //TOR warning
        if (leftLineConfidence_data.doubleValue == 0 &&
rightLineConfidence_data.doubleValue == 0) {
            if (tor_condition_check == false) {
                tor_condition_check = true;
                tor_initial_time = [[NSDate date] timeIntervalSince1970];
            } else {
                if ([[NSDate date] timeIntervalSince1970] - tor_initial_time >=4) {
                    curWarning = AutomateWarning_TOR;
                }
            }
        } else {
            tor_condition_check = false;
        }
        }
    } else {
        tor_condition_check = false;
    }


    if (prevWarning!=curWarning ) {
        prevWarning = curWarning;
        [_delegate onWarningDetected:curWarning];

    }

}

@end
```