Delphi Electronics & Safety

iCup

Software Detailed Design Document

**Prepared for Dataelements**

**Document: SDD-**10030170**-**<XXX>

**Prepared by: Johan Olsson**

**Date: 2017-01-02 Version:** <0.1>

**Revision** **History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Date** | **State** | **Author** | **Company/**  **Department** | **Change Description** |
| 0 | 2016-08-15 | Draft | Johan Olsson | Volvo Cars | Draft version |

States of this document: draft, reviewed, approved, final

**Table of Contents**

[1Introduction 2](#__RefHeading___Toc811_766363851)

[1.1Purpose 2](#__RefHeading___Toc813_766363851)

[1.2Scope 2](#__RefHeading___Toc815_766363851)

[1.3Definitions, acronyms, and abbreviations 2](#__RefHeading___Toc817_766363851)

[1.4References 3](#__RefHeading___Toc819_766363851)

[1.5Overview 3](#__RefHeading___Toc821_766363851)

[2Software structure and interfaces 5](#__RefHeading___Toc823_766363851)

[2.1Application interfaces 5](#__RefHeading___Toc825_766363851)

[2.1.2 Send Example 6](#__RefHeading___Toc827_766363851)

[2.1.3 Receive Example 6](#__RefHeading___Toc829_766363851)

[using namespace ApplicationDataElements; 6](#__RefHeading___Toc831_766363851)

[2.2 Class diagram 8](#__RefHeading___Toc833_766363851)

[2.3 Tools 8](#__RefHeading___Toc835_766363851)

[8](#__RefHeading___Toc2610_1900665739)

[3Dynamic behavior and algorithms 9](#__RefHeading___Toc837_766363851)

[4Resource consumption objectives 9](#__RefHeading___Toc839_766363851)

[5Test criteria 10](#__RefHeading___Toc841_766363851)

[6Open issues 10](#__RefHeading___Toc843_766363851)

[7Appendixes 10](#__RefHeading___Toc845_766363851)

# Introduction

## Purpose

This document is intended to be used as reference to the implementation of data elements and give developers an architectural and design overview.

## Scope

This document decribes the data elements framework in the ICup project. The data elements framework allows you to send and receive autosar(Flexray and LIN) signals on the MP processor.

## Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Explanation** |
| IHU | Infotainment Head Unit |
| MP | Media Processor |
| VIP | Vehicle Interface Processor |

## References

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Document Name** | **Date** | **Version** | **Location** |
|  |  |  |  |  |
|  |  |  |  |  |

## Overview

The Dataelements framework (sometimes also called Signals) is located in the ihu\_signals repository and is a part of the IHU SDK. Dataelements shall be used by applications with the need to send and receive autosar signals, both Flexray and Lin. Big parts of the framework is generated from the Com.arxml and Swc.arxml ECU-extracts received from Volvo Cars to guarantee up-to-date and correct signal definitions.

Dataelements provides the following functionality:

1. Send (autosar) signal
2. Receive signal
3. Subscribe to signal change
4. Type-safety for signals and signal content
5. Built in scaling/offsetting to always provide correct physical values (when applicable)

The below pictures gives an overview of the complete signaling package.

### 1.5.1 Flexray Signalling Overview



The Vehicle Interface Processor contains the AutoSAR stack and the physical connection to the flexray bus. For incoming signals the flexray signals are received by the IHU Software component in the VIP processor and transferred to the MP processor via an UART connection. On the Media Processor the VehicleSignalsManager reads from the UART and injects the signals to the Linux applications/daemons using the dataelements framework. For outgoing signals the message path is the same but in opposite order.

The protocol between the VIP and the MP that carries the autosar signals is called AVMP (Autosar VIP-MP Protocol). AVMP is transported on top of desip which then directly uses the uart. For details about the AVMP protocol see <https://delphisweden.atlassian.net/wiki/pages/viewpage.action?spaceKey=VI&title=VIP-MP+signaling+design> and/or the design document for Vehicle Signals Manager(VSM)

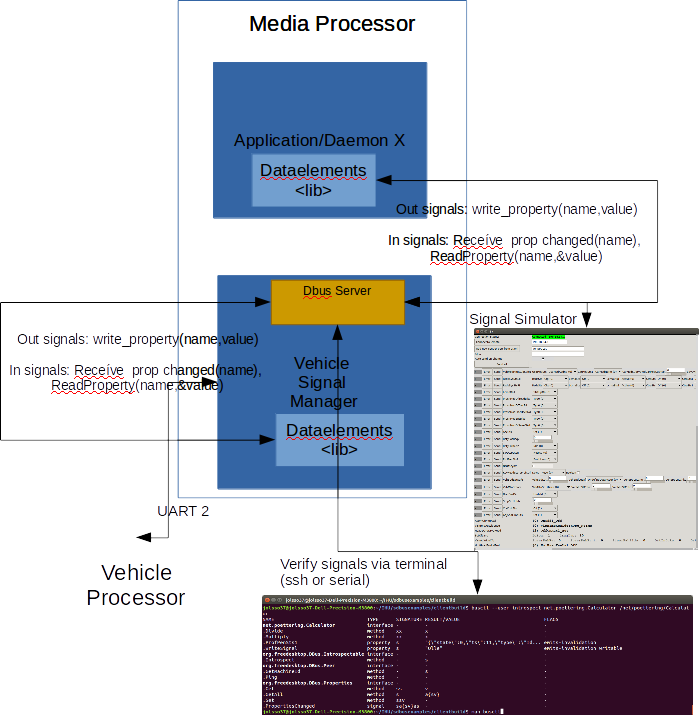
### 1.5.2 Code generation

To be able to handle the huge amount of signals in the IHU there is a need for a solid code generation chain that is using the Com & Swc ECU extracts. On the VIP the Vector tool chain is used to generate decoders and encoders from these extracts. Received signals in the VIP have to be encoded again, forwarded to the MP, decoded, optionally scaled etc and then provided to the applications via the Dataelement framework. This is a long chain and cannot be implemented with high quality without using code generation. Therefor also code generation has to be used for the communication between the VIP and MP and for the code within the MP.

The code generation process is as follows.

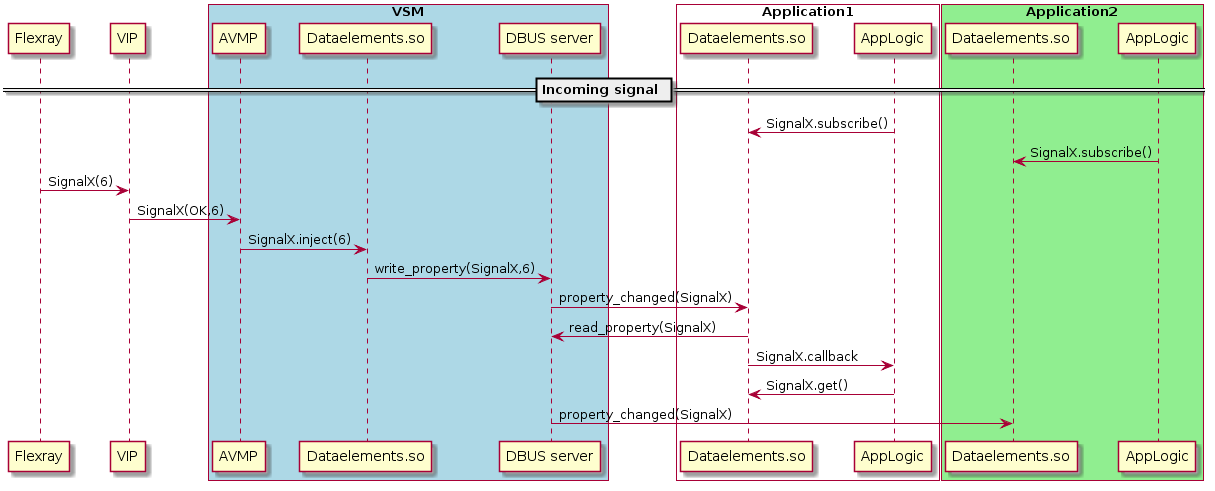


### 1.5.3 Flexray Signalling Overview Media Processor



The dataelements lib are used by all applications, daemons and plugins(libs) that want to send or receive a flexray signals. When sending out a signal the dataelements library converts the signal data represented by autosar types into a readable json format. And the other way around when receiving a signal. To send the signals to/from the vehicle signal manager (that communicates with the VIP) Linux DBUS communication is used. Each signal is represented as DBUS property. The applications/daemons do DBUS "add match" on the SignalsChanged DBUS signal and then read the the DBUS property that are named as the signal name and contains the signal data in JSON format. To send out the signal DBUS property write is used, which then updates the corresponding DBUS property in the DBUS server that resides as a seperate thread in the VSM daemon.

The part of Vehicle Signals Manager that sends/reads the signals from the UART is also using the dataelements lib when updating the signals in the DBUS properties. See the sequence diagram for a understanding of the way an in signal is distributed in the system.



The convertion to/from JSON format and the DBUS is hidden by the data elements framework and is not something the applications need to care about. The dataelement library checks at compile time that the autosar defined data types for the signals are correct (e.g. you can't send a string in a enum) and also that it is not possible to send an “in signal” and the other way around. The different signal data types and signal names are generated from the autosar definition file.

During development the developers also use different tools to monitor the signals that are sent out and received on the MP side. By this the application developer can test their vehicle function related functionality without having a connection to the physical flexray bus.

This document only describes the design of the dataelements framework, it does not include the detailed design of the related modules as VSM and VIP.

# Software structure and interfaces

## Application interfaces

To send and receive signals you use one of the classes:

ApplicationDataElement::DESender<T>  and ApplicationDataElement::DEReceiver<T>.

These are template classes. They shall be (template) instantiated with one of the available structs in "dataelements/gen\_dataelements.h". Each struct is autogenerated from the autosar definition file and represents a signal. Each struct inherits from InTag or OutTag or InternalTag. This means that the signal is an in-signal , out-signal , ihu-internal signal respectively.

In the body of the structs there is something like "using data\_elem\_type = <some\_type>;" . This shows the type of the data for a particular signal. The type can for example be a double, bool, int but it can also be an enum class or another struct. If the type is more "complicated" this type is declared in the file "dataelements/gen\_datatypes.h" .

### 2.1.2 Send Example

**using** **namespace** ApplicationDataElements;

// This is an out-signal where the data type is a struct: autosar::OnOffPen

// This declaration is most commonly put as an variable/attribute in your class

DESender<autosar::AccAdprSpdLimActv\_info> outSignal1;

autosar::OnOffPen dataForSignal;

dataForSignal.Sts = autosar::OnOff1::Off; // the data/struct has two fields: .Sts and .Pen

dataForSignal.Pen = autosar::IdPen::Prof4;

outSignal1.send(dataForSignal);

DESender<autosar::HdrestFoldReq2\_info> outSignal2; // This is an out-signal where the data type is a bool

outSignal2.send(**true**);

### 2.1.3 Receive Example

### **using** **namespace** ApplicationDataElements;

 // This declaration is most commonly put as an variable/attribute in your class

DEReceiver<KeySpdWarn\_info> anInSignal;

auto v1 = anInSignal.get();

**if** ( v1.isOk() ) // Always check the state before using the value

{ // The state is OK, lets go ahead and read the signal value

    autosar::Trig1 value = v1.value();

    :

    :

}

**else**

{ // The signal state is ERROR so you shall NOT read the value

    :

}

// You can also subscribe to signal changes (state and/or value change)

anInSignal.subscribe( [&]() {

    auto v2 = anInSignal.get();

**if** ( v2.isOk() ) // Always check the state before using the value

    { // The state is OK, lets go ahead and read the signal value

        autosar::Trig1 value = v2.value();

        :

    }

**else**

    { // The signal state is ERROR so you shall NOT read the value

        :

    }

});

/ You can also subscribe to signal changes (state and/or value change)

anInSignal.subscribe( [&]() {

    auto v2 = anInSignal.get();

**if** ( v2.isOk() ) // Always check the state before using the value

    { // The state is OK, lets go ahead and read the signal value

        autosar::Trig1 value = v2.value();

        :

    }

**else**

    { // The signal state is ERROR so you shall NOT read the value

        :

    }

});

## 2.2 Class diagram

### 

### 

### 2.3 Tools

There are different alternatives to view and inject signals in the MP processor:

**signaltrace**

If you have a terminal connection to the MP you can view the incoming and outgoing signals by using the signaltrace program. This program connects to the DBUS Server residing in the VSM process to read the current signal data.

Simulator tool

With this tool you can view and simulate different signals using a graphical user interface. This tool runs on your host and requires that you have development image loaded and a network connection to the target.

This tool connects to the DBUS server on target using a TCP/IP socket connection. It uses the python dbus library to set and read the dbus properties that contains the signal data.

### 

# Dynamic behavior and algorithms

TBD

# Resource consumption objectives

TBD

**Memory usage**

Table 1 displays the memory usage for the vehiclefunctions service

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Process Name** | **Code**  **[.text]** | **Data (static)**  **[.data + .bss]** | **Data**  **(stack)** | **Data**  **(heap)** | **Non-vol  [e.g. eMMC]** | **Size of exec** | **RAM disk** |
|  |  |  |  |  |  |  |  |

Table 1 Memory usage by vehiclefunctions(size in kB)

Table 1 displays the memory usage for the vehiclefunctions server modules.

|  |  |  |  |
| --- | --- | --- | --- |
| **File name** | **Code** | **Data** | **Non-vol  [e.g. EEPROM]** |
|  |  |  | **0** |
|  |  |  | **0** |
|  |  |  | **0** |
|  |  |  | **0** |
|  |  |  | **0** |
| **Total** | **0** | **0** | **0** |

Table 2 Memory usage by vehiclefunctions (size in bytes)

**CPU usage / Execution Time at start up and shutdown**

This chapter describes the CPU Usage and the execution time during the system start-up / shut down

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process Name** | **Average CPU usage [%]** | **Max CPU usage [%]** | **Start-up time** | **Shut-down time** |
| **vehiclefunctions** | **NA** | **NA** | **NA** | **NA** |

Table 3 CPU usage by vehiclefunctions

**Threads**

The dataelements framework spawns a separate thread for handling incoming messages to the dataelements framework. Otherwise the dataelements lib is executed in the context of the calling thread.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thread name** | **Priority** | **Execution time**  **(SOH / Priority Watchdog time-out)** | **Scheduling algorithm** | **Shared System resources** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **Total** |  |  |  |  |

Table 3 Example of occupied resource by vehiclefunctions threads

**Shared objects / libraries**

|  |  |
| --- | --- |
| **Library name** | **Linked (statically / dynamically)** |
|  | statically |
|  | dynamically |

Table 4 Example of shared objects used by vehiclefunctions

# Test criteria

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Unit, function, functionality, sequence of execution etc.** | **Testing inputs** | **Proposed code coverage** |
|  |  |  |  |
|  |  |  |  |

# Open issues

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Issue** | **Comments / Proposed Solution** | **Person responsible** | **Issue to be resolved by date** |
|  |  |  |  |  |
|  |  |  |  |  |

# Appendixes