UMRRA4 AUTOMOTIVE V1.0.0 USER INTERFACE

DATE:
January 22, 2024

USER INTERFACE NAME:
UMRRA4 AUTOMOTIVE

USER INTERFACE VERSION:
v1.0.0

s.m.s, smart microwave sensors GmbH
In den Waashainen 1
38108 Braunschweig
Germany

Phone: +49 531 39023-0
Fax: +49 531 39023-599
info@smartmicro.de
www.smartmicro.com

CONTENT

1 COMMUNICATION DATA STREAM SERVICE

With the communication data stream service smartmicro ports can be received as C++ objects with simplified access functions, which are generated by the user interface. Smartmicro ports are data buffers which contains data recorded by the radar data: e.g objects, statistics, statuses of device etc. Each port contains a generic port header, with a port description : version, id, size etc. Sometime ports also contains dynamic list of objects. In order to receive a port, a callback needs to be registered with the service. The callback will be carried out periodically every sensor cycle time.

Please note:

- This callback will be called in the context of a receiver thread, so the data needs to be copied and the callback must be released. Otherwise, the reception will be blocked.
- It is possible to use one callback function for several clients with the same port and same user interface, but it is not allowed to use one callback function for different ports or different user interfaces.

For more details please see the examples below. The following ports are supported.

1.1 COMTARGETLISTPORT PORT

Description:
To receive the port called "ComTargetListPort" from a specific client, please use the following registration interface:

```
#include <umrra4_automotive_v1_0_0/DataStreamService.h>

void ReceiveComTargetListPortClbk(IN std::shared_ptr<com::master::umrra4_automotive_v1_0_0::
    comtargetlistport::ComTargetListPort> comTargetListPort, com::types::ClientId clientId)
{
        // Getting members of  ComTargetListPort
    std::shared_ptr<com::master::umrra4_automotive_v1_0_0::comtargetlistport::GenericPortHeader>
        genericPortHeader =
                        comTargetListPort->GetGenericPortHeader();
    std::shared_ptr<com::master::umrra4_automotive_v1_0_0::comtargetlistport::StaticPortHeader>
        staticPortHeader =
                        comTargetListPort->GetStaticPortHeader();
    auto targetList = comTargetListPort->GetTargetList();
    // Getting members of  GenericPortHeader
    std::cout << "Variable PortId:"
              << genericPortHeader->GetPortId()
              << std::endl;
    std::cout << "Variable PortVersionMajor:"
              << genericPortHeader->GetPortVersionMajor()
              << std::endl;
    std::cout << "Variable PortVersionMinor:"
              << genericPortHeader->GetPortVersionMinor()
              << std::endl;
    std::cout << "Variable Timestamp:"
              << genericPortHeader->GetTimestamp()
              << std::endl;
    std::cout << "Variable PortSize:"
              << genericPortHeader->GetPortSize()
              << std::endl;
    std::cout << "Variable BodyEndianness:"
              << genericPortHeader->GetBodyEndianness()
              << std::endl;
    std::cout << "Variable PortIndex:"
              << genericPortHeader->GetPortIndex()
              << std::endl;
    std::cout << "Variable HeaderVersionMajor:"
              << genericPortHeader->GetHeaderVersionMajor()
              << std::endl;
    std::cout << "Variable HeaderVersionMinor:"
              << genericPortHeader->GetHeaderVersionMinor()
```

```
            << std::endl;
    // Getting members of  StaticPortHeader
    std::cout << "Variable CycleTime:"
            << staticPortHeader->GetCycleTime()
            << std::endl;
    std::cout << "Variable NumberOfTargets:"
            << staticPortHeader->GetNumberOfTargets()
            << std::endl;
    std::cout << "Variable AcquisitionTx:"
            << staticPortHeader->GetAcquisitionTx()
            << std::endl;
    std::cout << "Variable AcquisitionSweep:"
            << staticPortHeader->GetAcquisitionSweep()
            << std::endl;
    std::cout << "Variable AcquisitionCf:"
            << staticPortHeader->GetAcquisitionCf()
            << std::endl;
    std::cout << "Variable AcquisitionPrfIdx:"
            << staticPortHeader->GetAcquisitionPrfIdx()
            << std::endl;
    std::cout << "Variable UmambiguousSpeed:"
            << staticPortHeader->GetUmambiguousSpeed()
            << std::endl;
    std::cout << "Variable AcquisitionStart:"
            << staticPortHeader->GetAcquisitionStart()
            << std::endl;
    // Getting members of  Target
    for(auto& target : targetList)
    {
        std::cout << "Variable Range:"
                << target->GetRange()
                << std::endl;
        std::cout << "Variable SpeedRadial:"
                << target->GetSpeedRadial()
                << std::endl;
        std::cout << "Variable AzimuthAngle:"
                << target->GetAzimuthAngle()
                << std::endl;
        std::cout << "Variable ElevationAngle:"
                << target->GetElevationAngle()
                << std::endl;
        std::cout << "Variable RCS:"
                << target->GetRCS()
                << std::endl;
        std::cout << "Variable Power:"
                << target->GetPower()
                << std::endl;
        std::cout << "Variable TgtNoise:"
                << target->GetTgtNoise()
                << std::endl;
    }

}
    auto comDataStreamServ = com::master::umrra4_automotive_v1_0_0::DataStreamServiceIface::Get()
        ;
    ClientId clientIdA = 1024; // client id from sensor a
    ClientId clientIdB = 1025; // client id from sensor b
    ReceiveComTargetListPortCallback callback =
                        std::bind(&ReceiveComTargetListPortClbk,
                        std::placeholders::_1,
                        std::placeholders::_2);
    if(ERROR_CODE_OK != comDataStreamServ->RegisterComTargetListPortReceiveCallback(clientIdA,
        callback)
    {
        std::cout << "Failed to register ComTargetListPort port callback" << std::endl;
    }
    if(ERROR_CODE_OK != comDataStreamServ->RegisterComTargetListPortReceiveCallback(clientIdB,
```

```
        callback))
    {
        std::cout << "Failed to register ComTargetListPort port callback" << std::endl;
    }
```

1.2 DIAGNOSTICPORT PORT

Description:
To receive the port called "DiagnosticPort" from a specific client, please use the following registration interface:

```
#include <umrra4_automotive_v1_0_0/DataStreamService.h>

void ReceiveDiagnosticPortClbk(IN std::shared_ptr<com::master::umrra4_automotive_v1_0_0::
    diagnosticport::DiagnosticPort> diagnosticPort, com::types::ClientId clientId)
{
        // Getting members of  DiagnosticPort
    std::shared_ptr<com::master::umrra4_automotive_v1_0_0::diagnosticport::GenericPortHeader>
        genericPortHeader =
                        diagnosticPort->GetGenericPortHeader();
    std::shared_ptr<com::master::umrra4_automotive_v1_0_0::diagnosticport::StaticPortHeader>
        staticPortHeader =
                        diagnosticPort->GetStaticPortHeader();
    auto diagnosticObjectList = diagnosticPort->GetDiagnosticObjectList();
    // Getting members of  GenericPortHeader
    std::cout << "Variable PortId:"
              << genericPortHeader->GetPortId()
              << std::endl;
    std::cout << "Variable PortVersionMajor:"
              << genericPortHeader->GetPortVersionMajor()
              << std::endl;
    std::cout << "Variable PortVersionMinor:"
              << genericPortHeader->GetPortVersionMinor()
              << std::endl;
    std::cout << "Variable Timestamp:"
              << genericPortHeader->GetTimestamp()
              << std::endl;
    std::cout << "Variable PortSize:"
              << genericPortHeader->GetPortSize()
              << std::endl;
    std::cout << "Variable BodyEndianness:"
              << genericPortHeader->GetBodyEndianness()
              << std::endl;
    std::cout << "Variable PortIndex:"
              << genericPortHeader->GetPortIndex()
              << std::endl;
    std::cout << "Variable HeaderVersionMajor:"
              << genericPortHeader->GetHeaderVersionMajor()
              << std::endl;
    std::cout << "Variable HeaderVersionMinor:"
              << genericPortHeader->GetHeaderVersionMinor()
              << std::endl;
    // Getting members of  StaticPortHeader
    std::cout << "Variable numberOfDiagnostics:"
              << staticPortHeader->GetnumberOfDiagnostics()
              << std::endl;
    std::cout << "Variable summarizedStatus:"
              << staticPortHeader->GetsummarizedStatus()
              << std::endl;
    std::cout << "Variable timestampFormat:"
              << staticPortHeader->GettimestampFormat()
              << std::endl;
    std::cout << "Variable timestamp:"
              << staticPortHeader->Gettimestamp()
              << std::endl;
```

```
        // Getting members of  DiagnosticObject
        for(auto& diagnosticObject : diagnosticObjectList)
        {
            std::cout << "Variable id:"
                      << diagnosticObject->Getid()
                      << std::endl;
            std::cout << "Variable status:"
                      << diagnosticObject->Getstatus()
                      << std::endl;
            std::cout << "Variable dataType:"
                      << diagnosticObject->GetdataType()
                      << std::endl;
            std::cout << "Variable value:"
                      << diagnosticObject->Getvalue()
                      << std::endl;
        }

    }
        auto comDataStreamServ = com::master::umrra4_automotive_v1_0_0::DataStreamServiceIface::Get()
            ;
        ClientId clientIdA = 1024; // client id from sensor a
        ClientId clientIdB = 1025; // client id from sensor b
        ReceiveDiagnosticPortCallback callback =
                              std::bind(&ReceiveDiagnosticPortClbk,
                              std::placeholders::_1,
                              std::placeholders::_2);
        if(ERROR_CODE_OK != comDataStreamServ->RegisterDiagnosticPortReceiveCallback(clientIdA,
            callback)
        {
            std::cout << "Failed to register DiagnosticPort port callback" << std::endl;
        }
        if(ERROR_CODE_OK != comDataStreamServ->RegisterDiagnosticPortReceiveCallback(clientIdB,
            callback))
        {
            std::cout << "Failed to register DiagnosticPort port callback" << std::endl;
        }
```

For a more detailed API description, please see Appendix A.

2 LEGAL DISCLAIMER NOTICE

All products, product specifications and data in this document may be subject to change without notice to improve reliability, function or otherwise. Not all products and/or product features may be available in all countries and regions. For legal reasons features may be deleted from products or smartmicro may refuse to offer products. Statements, technical information and recommendations contained herein are believed to be accurate as of the stated date. smartmicro disclaims any and all liability for any errors, inaccuracies or incompleteness contained in this document or in any other disclosure relating to the product.

To the extent permitted by applicable law, smartmicro disclaims (i) any and all liability arising out of the application or use of the product or the data contained herein, (ii) any and all liability of damages exceeding direct damages, including - without limitation - indirect, consequential or incidental damages, and (iii) any and all implied warranties, including warranties of the suitability of the product for particular purposes.

Statements regarding the suitability of products for certain types of applications are based on smartmicro's knowledge of typical requirements that are often placed on smartmicro products in generic/general applications. Statements about the suitability of products for a particular/specific application, however, are not binding. It is the customer's/user's responsibility to validate that the product with the specifications described is suitable for use in the particular/specific application. Parameters and the performance of products may deviate from statements made herein due to particular/specific applications and/or surroundings. Therefore, it is important that the customer/user has thoroughly tested the products and has understood the performance and limitations of the products before installing them for final applications or before their commercialization. Although products are well optimized to be used for the intended applications stated, it must also be understood by the customer/user that the detection probability may not be 100% and that the false alarm rate may not be zero.

The information provided, relates only to the specifically designated product and may not be applicable when the product is used in combination with other materials or in any process not defined herein. All operating parameters, including typical parameters, must be validated for each application by the customer's/user's technical experts. Customers using or selling smartmicro products for use in an application which is not expressly indicated do so at their own risk. This document does not expand or otherwise modify smartmicro's terms and conditions of purchase, including but not being limited to the warranty. Except as expressly indicated in writing by smartmicro, the products are not designed for use in medical, lifesaving or life-sustaining applications or for any other application in which the failure of the product could result in personal injury or death.

No license expressed or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document or by any conduct of smartmicro. Product names and markings noted herein may be trademarks of their respective owners.

Please note that the application of the product may be subject to standards or other regulations that may vary from country to country. smartmicro does not guarantee that the use of products in the applications described herein will comply with such regulations in any country. It is the customer's/user's responsibility to ensure that the use and incorporation of products comply with regulatory requirements of their markets.

If any provision of this disclaimer is, or is found to be, void or unenforceable under applicable law, it will not affect the validity or enforceability of the other provisions of this disclaimer.

A COMMUNICATION DATA SERVICE API

A **ComTargetListPort**
Description: Provides a list of radar targets.
The object ComTargetListPort provides the following APIs:

```
std::shared_ptr<com::master::umrra4_automotive_v1_0_0::GenericPortHeader>
    GetGenericPortHeader() const;
```

Returns pointer to GenericPortHeader object, whose access functions are described below:
A **GenericPortHeader**
Description: No description available
The object GenericPortHeader provides the following APIs:

```
uint32_t GetPortId() const;
```

Returns value of PortId of uint32_t data type.
PortId - Port identification number.

```
int16_t GetPortVersionMajor() const;
```

Returns value of PortVersionMajor of int16_t data type.
PortVersionMajor - Major version of the port API.

```
int16_t GetPortVersionMinor() const;
```

Returns value of PortVersionMinor of int16_t data type.
PortVersionMinor - Minor version of the port API.

```
uint64_t GetTimestamp() const;
```

Returns value of Timestamp of uint64_t data type.
Timestamp - Time of creation of the port.

```
uint32_t GetPortSize() const;
```

Returns value of PortSize of uint32_t data type.
PortSize - Size of the port including this header.

```
uint8_t GetBodyEndianness() const;
```

Returns value of BodyEndianness of uint8_t data type.
BodyEndianness - Endianness of the sensor system.Please notice that the data will be presented in the host endianness, and should not be reversed.

```
uint8_t GetPortIndex() const;
```

Returns value of PortIndex of uint8_t data type.
PortIndex - The port index is used for multiple occurance of ports with the same port identifier.

```
uint8_t GetHeaderVersionMajor() const;
```

Returns value of HeaderVersionMajor of uint8_t data type.
HeaderVersionMajor - Major version of the generic port API.

```
uint8_t GetHeaderVersionMinor() const;
```

Returns value of HeaderVersionMinor of uint8_t data type.
HeaderVersionMinor - Minor version of the generic port API.

```
std::shared_ptr<com::master::umrra4_automotive_v1_0_0::StaticPortHeader> GetStaticPortHeader
    () const;
```

Returns pointer to StaticPortHeader object, whose access functions are described below:
A **StaticPortHeader**
Description: No description available
The object StaticPortHeader provides the following APIs:

```
uint16_t GetnumberOfDiagnostics() const;
```

Returns value of numberOfDiagnostics of uint16_t data type.
numberOfDiagnostics - Number of diagnostics that are in the dynamic part of the port.

```
uint8_t GetsummarizedStatus() const;
```

Returns value of summarizedStatus of uint8_t data type.
summarizedStatus - The Summarized Status field is set to the most critical diagnostic status that is listed in the dynamic part of the port.

```
uint8_t GettimestampFormat() const;
```

Returns value of timestampFormat of uint8_t data type.
timestampFormat - 0 = No timestamp is set, 1 = Raw timestamp in microseconds, 2 = NTP format

```
uint64_t Gettimestamp() const;
```

Returns value of timestamp of uint64_t data type.
timestamp - Timestamp when the data was collected

```
const std::vector<std::shared_ptr<com::master::umrra4_automotive_v1_0_0::Target>>&
    GetTargetList() const;
```

Returns pointer to array of Target objects, whose access functions are described below:
A **Target**
Description: Represents a single target
The object Target provides the following APIs:

```
float32_t GetRange() const;
```

Returns value of Range of float32_t data type.
Range - Target range

```
float32_t GetSpeedRadial() const;
```

Returns value of SpeedRadial of float32_t data type.
SpeedRadial - Target speed

```
float32_t GetAzimuthAngle() const;
```

Returns value of AzimuthAngle of float32_t data type.
AzimuthAngle - Azimuth angle of the target

```
float32_t GetElevationAngle() const;
```

Returns value of ElevationAngle of float32_t data type.
ElevationAngle - Elevation angle of the target

```
float32_t GetRCS() const;
```

Returns value of RCS of float32_t data type.
RCS - The Radar Cross Section of the target

```
float32_t GetPower() const;
```

Returns value of Power of float32_t data type.
Power - Amplitude of the target

```
float32_t GetTgtNoise() const;
```

Returns value of TgtNoise of float32_t data type.
TgtNoise - Noise of the target

A **DiagnosticPort**
Description: Provide Diagnostic Values
The object DiagnosticPort provides the following APIs:

```
std::shared_ptr<com::master::umrra4_automotive_v1_0_0::GenericPortHeader>
    GetGenericPortHeader() const;
```

Returns pointer to GenericPortHeader object, whose access functions are described below:
A **GenericPortHeader**
Description: No description available
The object GenericPortHeader provides the following APIs:

```
uint32_t GetPortId() const;
```

Returns value of PortId of uint32_t data type.
PortId - Port identification number.

```
int16_t GetPortVersionMajor() const;
```

Returns value of PortVersionMajor of int16_t data type.
PortVersionMajor - Major version of the port API.

```
int16_t GetPortVersionMinor() const;
```

Returns value of PortVersionMinor of int16_t data type.
PortVersionMinor - Minor version of the port API.

```
uint64_t GetTimestamp() const;
```

Returns value of Timestamp of uint64_t data type.
Timestamp - Time of creation of the port.

```
uint32_t GetPortSize() const;
```

Returns value of PortSize of uint32_t data type.
PortSize - Size of the port including this header.

```
uint8_t GetBodyEndianness() const;
```

Returns value of BodyEndianness of uint8_t data type.
BodyEndianness - Endianness of the sensor system.Please notice that the data will be presented in the host endianness, and should not be reversed.

```
uint8_t GetPortIndex() const;
```

Returns value of PortIndex of uint8_t data type.
PortIndex - The port index is used for multiple occurance of ports with the same port identifier.

```
uint8_t GetHeaderVersionMajor() const;
```

Returns value of HeaderVersionMajor of uint8_t data type.
HeaderVersionMajor - Major version of the generic port API.

```
uint8_t GetHeaderVersionMinor() const;
```

Returns value of HeaderVersionMinor of uint8_t data type.
HeaderVersionMinor - Minor version of the generic port API.

```
std::shared_ptr<com::master::umrra4_automotive_v1_0_0::StaticPortHeader> GetStaticPortHeader
    () const;
```

Returns pointer to StaticPortHeader object, whose access functions are described below:
A **StaticPortHeader**
Description: No description available
The object StaticPortHeader provides the following APIs:

```
uint16_t GetnumberOfDiagnostics() const;
```

Returns value of numberOfDiagnostics of uint16_t data type.
numberOfDiagnostics - Number of diagnostics that are in the dynamic part of the port.

```
uint8_t GetsummarizedStatus() const;
```

Returns value of summarizedStatus of uint8_t data type.
summarizedStatus - The Summarized Status field is set to the most critical diagnostic status that is listed in the dynamic part of the port.

```
uint8_t GettimestampFormat() const;
```

Returns value of timestampFormat of uint8_t data type.
timestampFormat - 0 = No timestamp is set, 1 = Raw timestamp in microseconds, 2 = NTP format

```
uint64_t Gettimestamp() const;
```

Returns value of timestamp of uint64_t data type.
timestamp - Timestamp when the data was collected

```
const std::vector<std::shared_ptr<com::master::umrra4_automotive_v1_0_0::DiagnosticObject>>&
    GetDiagnosticObjectList() const;
```

Returns pointer to array of DiagnosticObject objects, whose access functions are described below:
A **DiagnosticObject**
Description: Collection of Diagnostic Outputs
The object DiagnosticObject provides the following APIs:

```
uint16_t Getid() const;
```

Returns value of id of uint16_t data type.
id - Unique id for each diagnostic function

```
uint8_t Getstatus() const;
```

Returns value of status of uint8_t data type.
status - Status of the diagnostic function

```
uint8_t GetdataType() const;
```

Returns value of dataType of uint8_t data type.
dataType - Data type of the value field

```
uint64_t Getvalue() const;
```

Returns value of value of uint64_t data type.
value - Value set by a diagnostic function

B USER INTERFACE INSTRUCTIONS UMRRA4 AUTOMOTIVE VERSION 1.0.0

B.1 Parameter Section auto_interface_0dim

Automotive user interface 0dimensional parameters

Parameter Name	center_frequency_idx
Description	Index of center frequency
Data Type	u8
Dimensions	None
Access	RW
Default	1
Min	0
Max	3

Parameter Name	frequency_sweep_idx
Description	Index of sweep: 0=X medium range, 1=X long range, 2=short range
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	2

Parameter Name	range_toggle_mode
Description	Automatic toggle of range: 0=off, 1=X long-X medium, 2=X long-short, 3=X medium-short
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	3

Parameter Name	prf_selector_manual
Description	0 = PRF switching active, 1 = PRF index given in prf_selector_index is used
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	1

Parameter Name	prf_set_selector
Description	Select PRF set index (in manual or automatic PRF mode)
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	1

Parameter Name	prf_manual_value_idx
Description	In manual PRF mode only: use nth element of selected set
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	2

Parameter Name	detection_sensitivity
Description	Detection sensitivity: 0=low, 1=normal, 2=high
Data Type	u8
Dimensions	None
Access	RW
Default	1
Min	0
Max	2

Parameter Name	tv_min_speed_sweep_idx_0
Description	Target Validation: minimum speed of target at first sweep
Data Type	f32
Dimensions	None
Access	RW
Default	-112.0
Min	-112.0
Max	56.0

Parameter Name	tv_min_speed_sweep_idx_1
Description	Target Validation: minimum speed of target at second sweep
Data Type	f32
Dimensions	None
Access	RW
Default	-112.0
Min	-112.0
Max	56.0

Parameter Name	tv_min_speed_sweep_idx_2
Description	Target Validation: minimum speed of target at third sweep
Data Type	f32
Dimensions	None
Access	RW
Default	-112.0
Min	-112.0
Max	56.0

Parameter Name	tv_max_speed_sweep_idx_0
Description	Target Validation: maximum speed of target at first sweep
Data Type	f32
Dimensions	None
Access	RW
Default	56.0
Min	-112.0
Max	56.0

Parameter Name	tv_max_speed_sweep_idx_1
Description	Target Validation: maximum speed of target at second sweep
Data Type	f32
Dimensions	None
Access	RW
Default	56.0
Min	-112.0
Max	56.0

Parameter Name	tv_max_speed_sweep_idx_2
Description	Target Validation: maximum speed of target at third sweep
Data Type	f32
Dimensions	None
Access	RW
Default	56.0
Min	-112.0
Max	56.0

Parameter Name	output_control_target_list_can
Description	send raw targets via CAN, 0 = disabled, 1 = enabled
Data Type	u8
Dimensions	None
Access	RW
Default	1
Min	0
Max	1

Parameter Name	output_control_object_list_can
Description	send objects via CAN, 0 = disabled, 1 = enabled
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	0

Parameter Name	output_control_target_list_eth
Description	send raw targets via Ethernet, 0 = disabled, 1 = enabled
Data Type	u8
Dimensions	None
Access	RW
Default	1
Min	0
Max	1

Parameter Name	output_control_object_list_eth
Description	send objects via Ethernet, 0 = disabled, 1 = enabled
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	0

Parameter Name	output_control_diagnostic_eth
Description	send diagnostic port via Ethernet, 0 = disabled, 1 = enabled
Data Type	u8
Dimensions	None
Access	RW
Default	1
Min	0
Max	1

Parameter Name	ip_source_address
Description	IP source address (32bit)
Data Type	u32
Dimensions	None
Access	RW
Default	3232238347

Parameter Name	subnet_mask
Description	Subnet mask (32bit)
Data Type	u32
Dimensions	None
Access	RW
Default	4294967040

Parameter Name	ip_dest_address
Description	IP destination address (32bit)
Data Type	u32
Dimensions	None
Access	RW
Default	3232238353

Parameter Name	ip_dest_port
Description	IP destination port
Data Type	u16
Dimensions	None
Access	RW
Default	55555

Parameter Name	mc_dest_address
Description	Multicast destination address for Alive (32bit)
Data Type	u32
Dimensions	None
Access	RW
Default	4019191808
Min	3758096384
Max	4026531839

Parameter Name	mc_port
Description	Multicast destination port for Alive
Data Type	u16
Dimensions	None
Access	RW
Default	60000
Min	1

Parameter Name	sync_mode
Description	(Master+Slave config) 0=off, 1=master, 2=slave
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	2

Parameter Name	sync_slave_identifier
Description	(Slave config) Unique Sync Slave Identifier, ignored on master (always 0)
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	7

Parameter Name	sync_group_identifier
Description	(Slave config) Sync Group Identifier, ignored on master (always 0)
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	1

Parameter Name	sync_nof_devices_1st_group
Description	(Master config) Number of synced devices (incl. master) in first group, ignored on slave
Data Type	u8
Dimensions	None
Access	RW
Default	1
Min	1
Max	8

Parameter Name	sync_nof_devices_2nd_group
Description	(Master config) Number of synced devices in second group, ignored on slave
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	7

Parameter Name	sync_interface
Description	(Master+Slave config) interface for sensor sync, 1=can,2=ethernet
Data Type	u8
Dimensions	None
Access	RW
Default	2
Min	1
Max	2

Parameter Name	time_sync_mode
Description	(Time Sync: Master+Slave config) 0=off, 1=master, 2=slave
Data Type	u8
Dimensions	None
Access	RW
Default	0
Min	0
Max	2

Parameter Name	time_sync_nof_devices
Description	(Time Sync: Master config) Number of time synced devices (incl. master), ignored on slave
Data Type	u8
Dimensions	None
Access	RW
Default	1
Min	1
Max	8

B.2 Status Section auto_interface

customer status section

Status Name	auto_interface_version_major
Description	Automotive interface version major. Increased, if new version is not totally backward compatible.
Data Type	u32
Dimensions	None
Access	R

Status Name	auto_interface_version_minor
Description	Automotive interface version minor. Increased, if parameters or statuses are changed or added. The new version is still backward compatible.
Data Type	u32
Dimensions	None
Access	R

Status Name	sw_generation
Description	Software Version generation
Data Type	u16
Dimensions	None
Access	R

Status Name	sw_version_major
Description	Software Version major
Data Type	u16
Dimensions	None
Access	R

Status Name	sw_version_minor
Description	Software Version minor
Data Type	u16
Dimensions	None
Access	R

Status Name	sw_version_patch
Description	Software Version patch
Data Type	u16
Dimensions	None
Access	R

Status Name	customer_id
Description	Customer Identifier
Data Type	u16
Dimensions	None
Access	R

Status Name	product_serial
Description	32Bit product id serial
Data Type	u32
Dimensions	None
Access	R

Status Name	product_gen
Description	product generation
Data Type	u32
Dimensions	None
Access	R

Status Name	product_mod_high
Description	product modification high
Data Type	u32
Dimensions	None
Access	R

Status Name	product_mod_low
Description	product modification low
Data Type	u32
Dimensions	None
Access	R

Status Name	product_rev
Description	product revision
Data Type	u32
Dimensions	None
Access	R

B.3 Command Section auto_interface_command

Maintain compatible section 1000 commands

Command Name	comp_fsm_core0_opmode
Description	Select top level FSM operation mode (3078.1)

Command Name	comp_eeprom_ctrl_factory_reset
Description	Performs factory reset (3102.4)

Command Name	comp_sensor_reset
Description	Reset command which starts from BIOS (if available) or bootloader (3074.1)

Command Name	comp_pdi_requestor_can
Description	Send PDI data to client (3076.1)

Command Name	comp_eeprom_ctrl_save_param_sec
Description	Save the parameter inside the EEPROM. (3102.3)

Command Name	comp_eeprom_ctrl_reset_param_sec
Description	Restore default values in RAM. EEPROM content is not changed. (3102.2)

Command Name	comp_eeprom_ctrl_default_param_sec
Description	Restore default values in RAM and EEPROM. (3102.1)

Command Name	comp_timebase_set_seconds_val
Description	Set SECONDS value of NTP UTC timestamp

Command Name	comp_timebase_set_frac_seconds_val
Description	Set FRACTION_SECONDS value of NTP UTC timestamp