# Sent_30_Icu

## Technical Reference

Project Release

Version 1.0.4


| | |
|---|---|
| Authors | Denis Althapp |
| Status | Not Released |

# Document Information

## History

| Author | Date | Version | Remarks |
|---|---|---|---|
| virdea | [2015-06-22] | 0.1.0 | Initial version |
| virdea | [2015-06-23] | 1.0.0 | Ready for project release |
| virdea | [2015-07-13] | 1.0.1 | Small graphical changes |
| virdea | [2015-07-31] | 1.0.2 | Updated architecture with notification functions and new APIs |
| virdea | [2015-08-05] | 1.0.3 | Added InitMemory API |
| virdea | [2015-10-15] | 1.0.4 | Added diagnostic APIs |

## Reference Documents

| No. | Source | Title | Version |
|---|---|---|---|
| [1] | SAE | J2716 SENT Specification | JAN2010 |
| [2] | NXP | KMA215 | [02/14] |
| [3] | Vector | CT_Sent_30_Icu | 0.0.1 |
| [4] | AUTOSAR | AUTOSAR_SWS_ICU_Driver.pdf | V3.3.0 |
| [5] | AUTOSAR | AUTOSAR_SWS_DefaultErrorTracer.pdf | V4.2.1 |

**Caution**
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

## Contents

## Illustrations



## Tables

# 1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

| Component Version | New Features |
|---|---|
| [01.00.02] | Version ready for project release |
| [01.00.03] | Added notification functions and APIs for requesting SENT data |
| | |

Table 1-1 Component history

# 2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module Sent_30_Icu.

| | | |
|---|---|---|
| **Supported AUTOSAR Release*:** | 3.X, 4.X | |
| **Supported Configuration Variants:** | <pre-compile> | |
| **Vendor ID:** | SENT_30_ICU_VENDOR_ID | 30 decimal<br>(= Vector-Informatik, according to HIS) |
| **Module ID:** | SENT_30_ICU_MODULE_ID | 0x8001 decimal |

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The SENT (Single Edge Nibble Transmission) protocol is a non-expensive solution for transmitting sensor data to the ECU. It requires 3 wires: VCC, GND and SIG.

SENT data is transmitted in 4 bit units (= 1 nibble). The nibble data values are dependent on the interval between two falling edges on the SIG line. Each SENT frame consists of:

- 1 calibration pulse to normalize the following pulses
- 1 communication nibble for application data/slow channel messages
- X data nibbles
- 1 CRC nibble
- (1 optional pause pulse)

The software component Sent_30_Icu is a handler for providing this SENT sensor data to the application layer.

## 2.1 Architecture Overview

The following figure shows where the Sent_30_Icu is located in the AUTOSAR architecture.

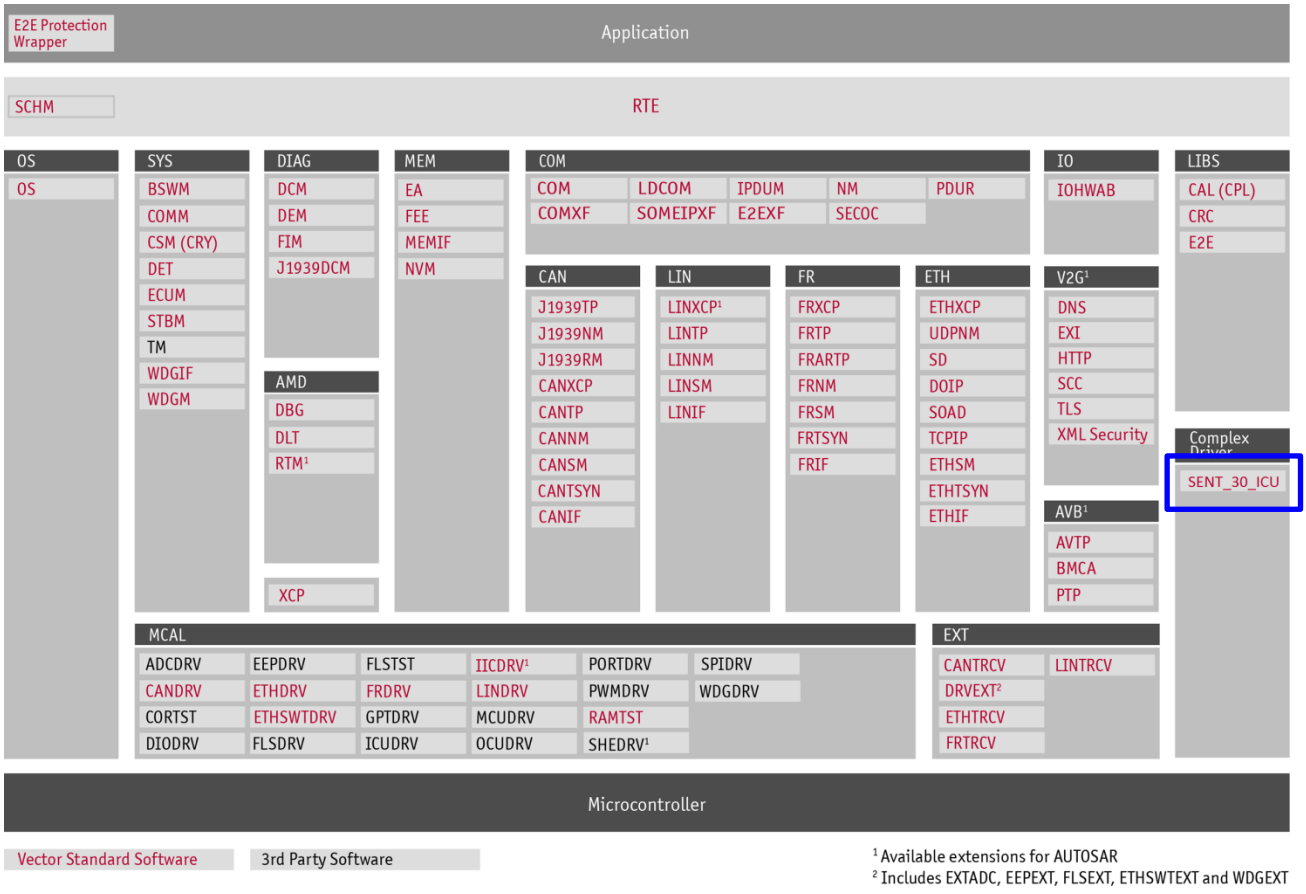


Figure 2-1 AUTOSAR 4.2 Architecture Overview

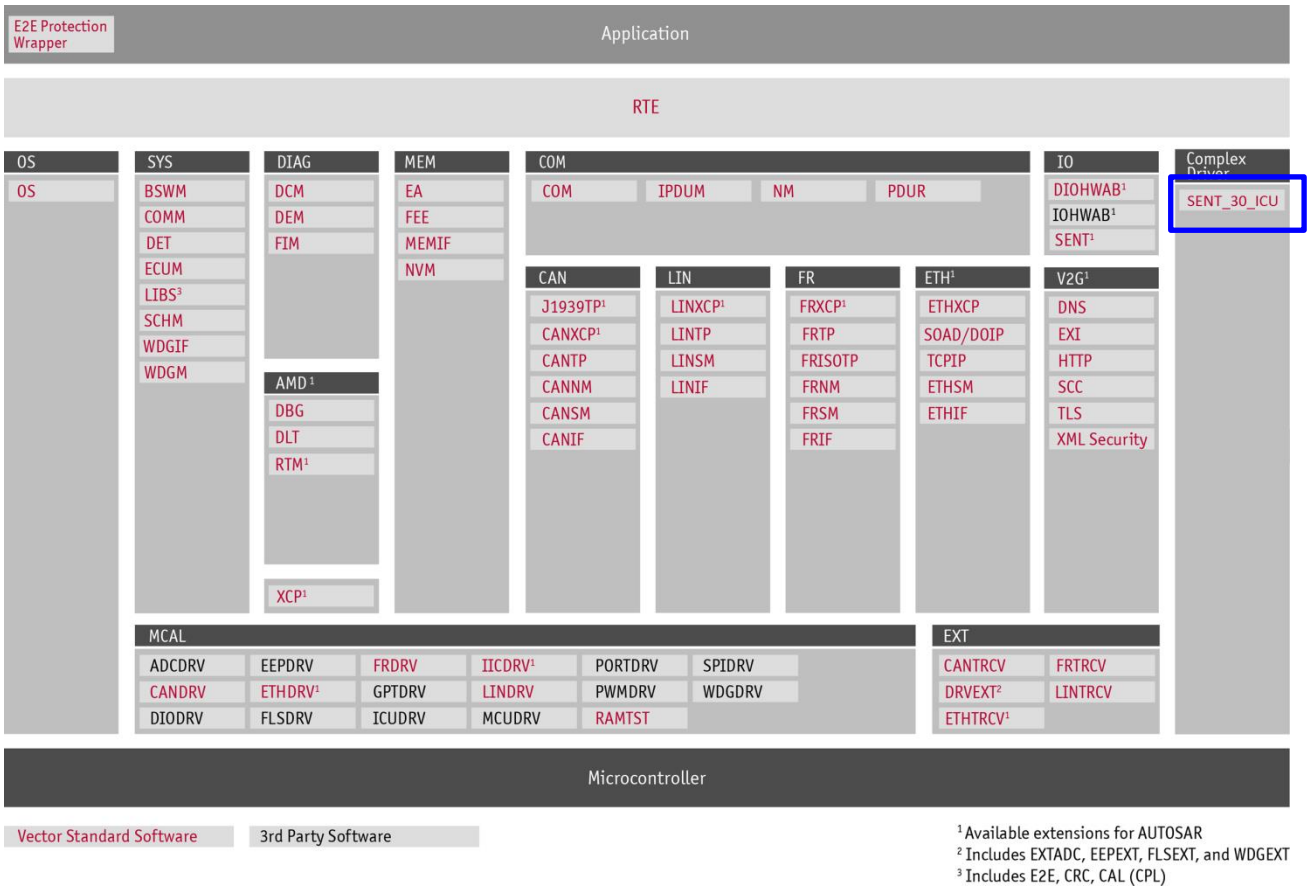


Figure 2-2 AUTOSAR 3.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the Sent_30_Icu. These interfaces are described in chapter 6.

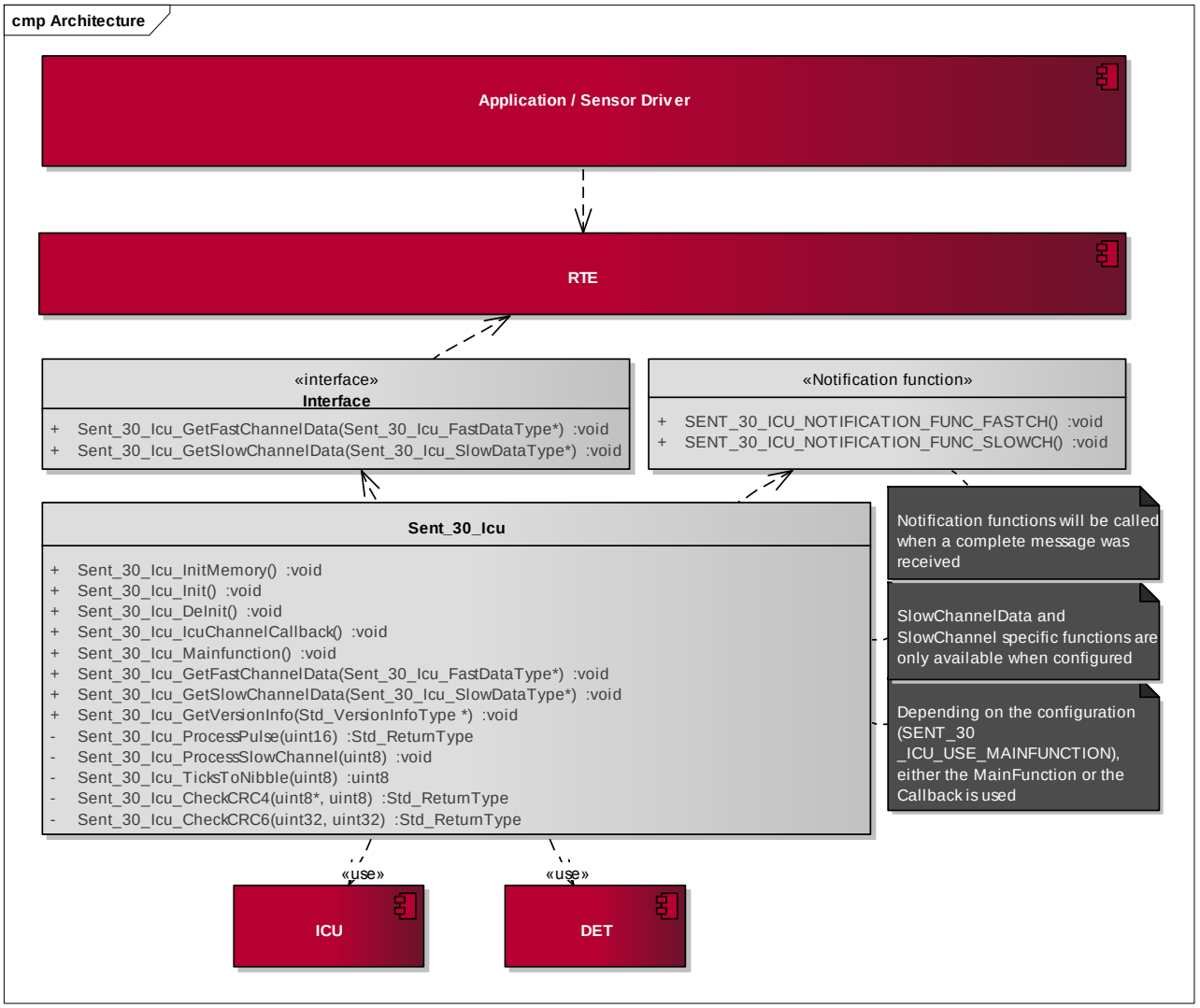


Figure 2-3 Interfaces to adjacent modules of the Sent_30_Icu

# 3 Functional Description

## 3.1 Use cases

The following use cases have been defined for the Sent_30_Icu:

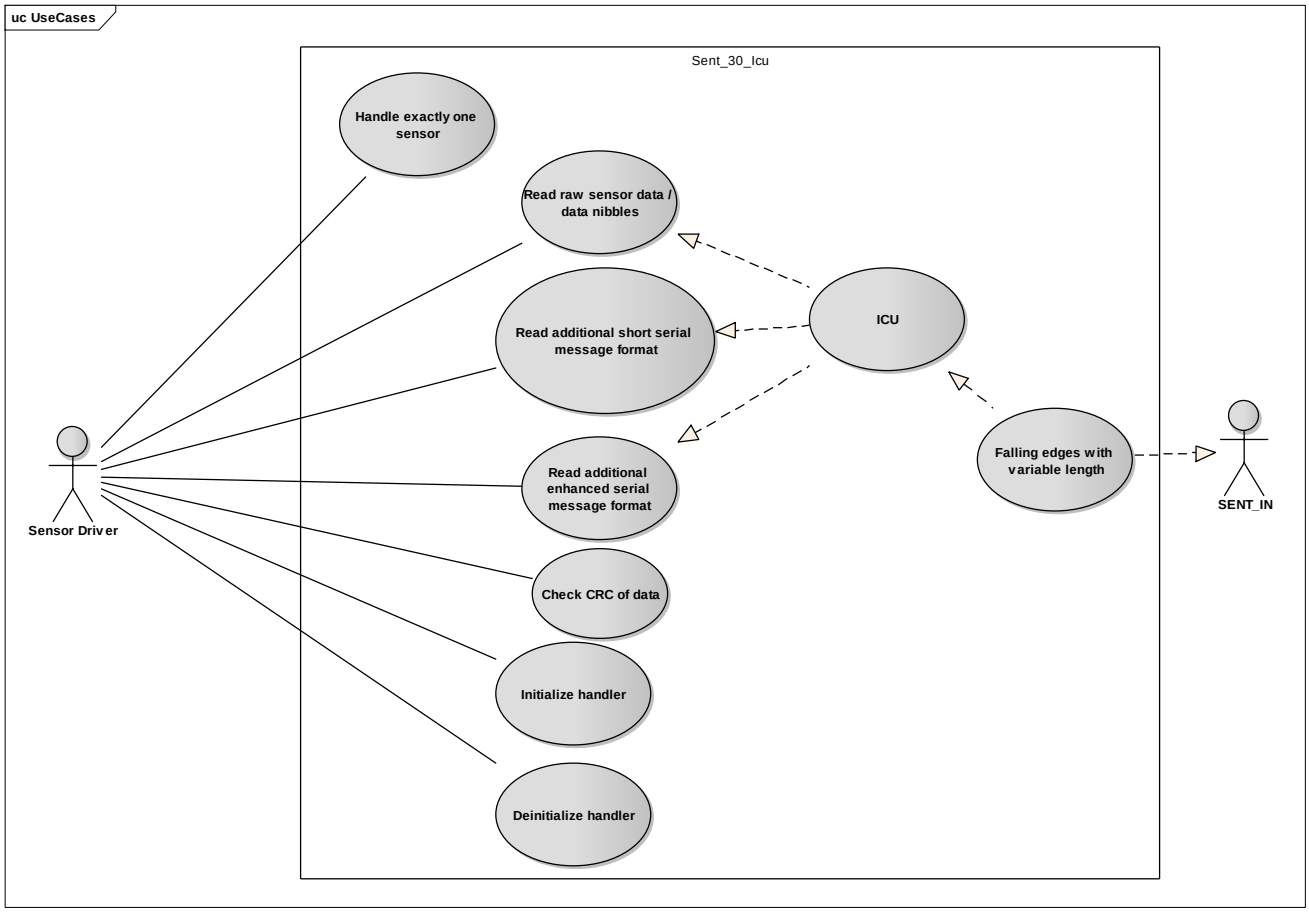


Figure 3-1 Defined Use-cases for the Sent_30_Icu

## 3.2 Features

The Sent_30_Icu component provides the following features:

- Support for one single SENT channel
- Provides SENT sensor data to the application layer
- Pre-compile time configuration to support any SENT sensor mode (variable tick times, data nibbles, pause pulse on/off)
- Support for short serial messages and enhanced serial messages
- Two different processing modes: MainFunction and IcuCallback
- Checks 4Bit-CRC and 6Bit-CRC of messages

## 3.3 Limitations

The Sent_30_Icu doesn't support ECU to sensor communication.

The component provides only the raw sensor data without metadata and doesn't align this data depending on the sensor data format (e.g. A1, A.3, H.3). This has to be done by the application.

To see the test cases for this release, see [3].

## 3.4 Required knowledge

The component measures falling edges of the SENT channel by ICU timestamping. It creates a delta between two timestamps to get the actual pulse length. The pulse length is then evaluated and stored. Once all pulses of a frame are received, the frame will be validated and (if configured) the slow channel nibble is further evaluated.

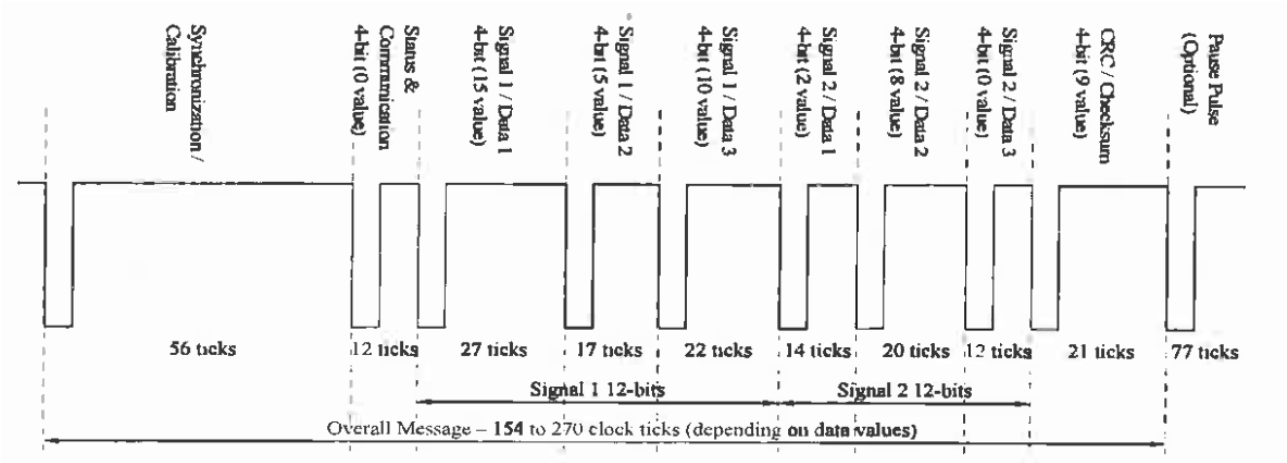


Figure 3-2 Typical SENT frame (see [1]) – the nibble values are dependent on the delta between falling edges

**Caution**

The Sent_30_Icu is a highly timing dependent component and therefore it's important to correctly configure the tick times depending on the ECU and SENT sensor or else the calibration pulses won't be detected. Once the calibration pulses are within detection range, further pulses will be normalized to this pulse length and the receiving should work.

The SENT protocol specifies also so called serial messages (further referenced as slow channel messages). Such messages are transferred via 2 bits of the "Status & Communication"-nibble (the other 2 bits are reserved for sensor specific application usage). Every 2 bits of slow-channel data requires one fast-channel frame. Therefore, for one slow-channel frame, a number of successive fast-channel frames (without any errors) are required:

- 16 for a short serial message
- 18 for an enhanced serial message

## 3.5 Behavior

### 3.5.1 Initialization

The initialization function of the Sent_30_Icu pre-calculates timing thresholds (to save time in further cyclic function calls) and also starts the ICU timestamping. It also starts the state machines and sets the module to initialized.

### 3.5.2 Processing of timestamps

The component can be configured to use either a cyclic mainfunction or the ICU callback notification to process the measured timestamps.

**Note**

The more efficient process mode depends on whether slow-channel messages are used or not. Slow-channel messages require several successive fast-channel messages to be successfully received. Therefore it’s recommended to use the callback method, because this ensures that no timestamp is missed.

When using the mainfunction method, the ICU-Buffer has to be large enough that in between two calls the ICU doesn’t overwrite unprocessed stamps (resulting in a very large buffer – 256*16uint16 for 10ms cycle time recommended = 4kB RAM).

### 3.5.2.1 MainFunction

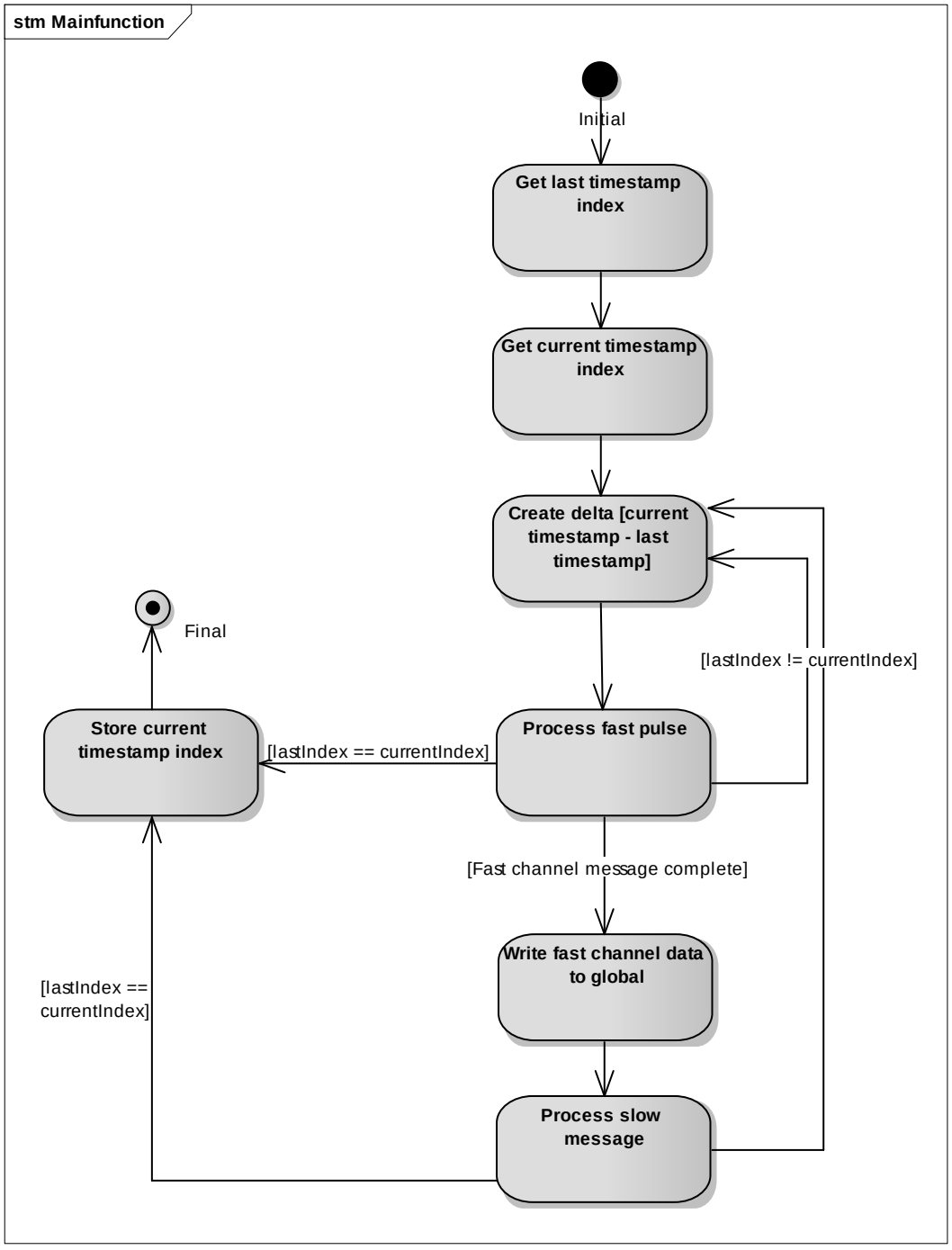


Figure 3-3 MainFunction processing

The mainfunction iterates over all timestamp indexes which haven't been processed since the last function call. Each delta is then separately processed and, upon a complete message, written to a global.

### 3.5.2.2 IcuCallback

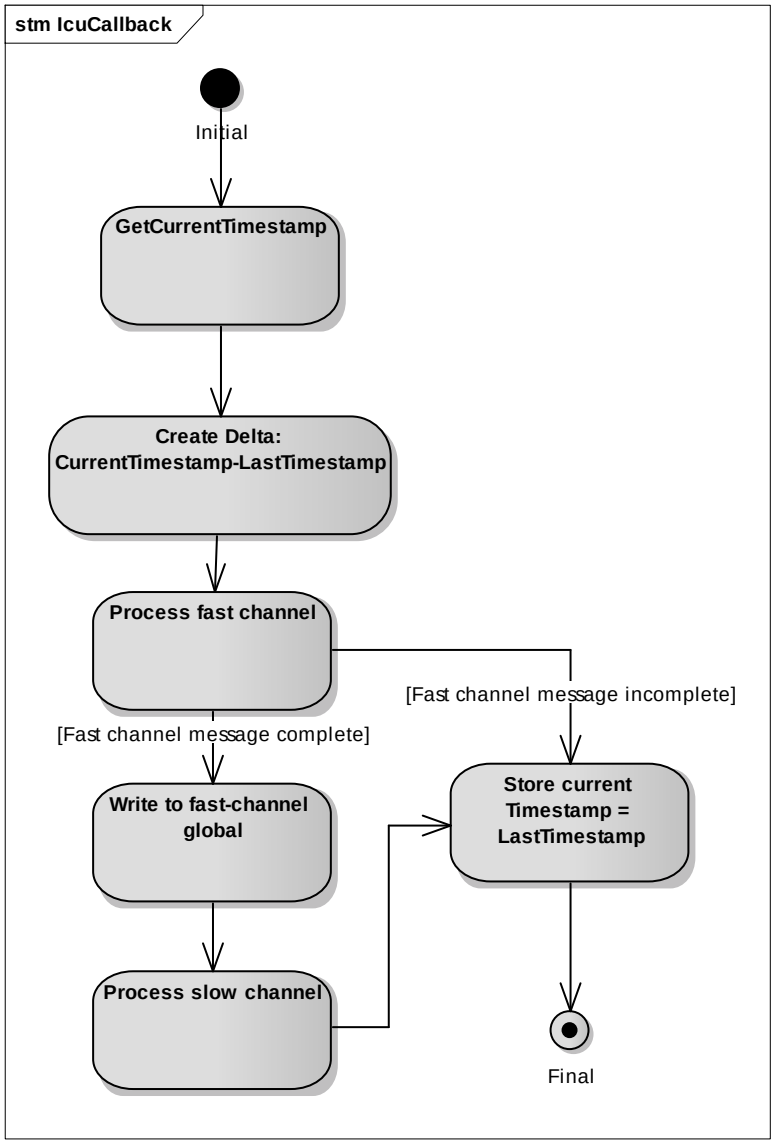


Figure 3-4 ICU callback processing

Each callback processes exactly one delta between the last timestamp and the current timestamp and feeds the length into the fast channel function.

### 3.5.3 Fast channel processing

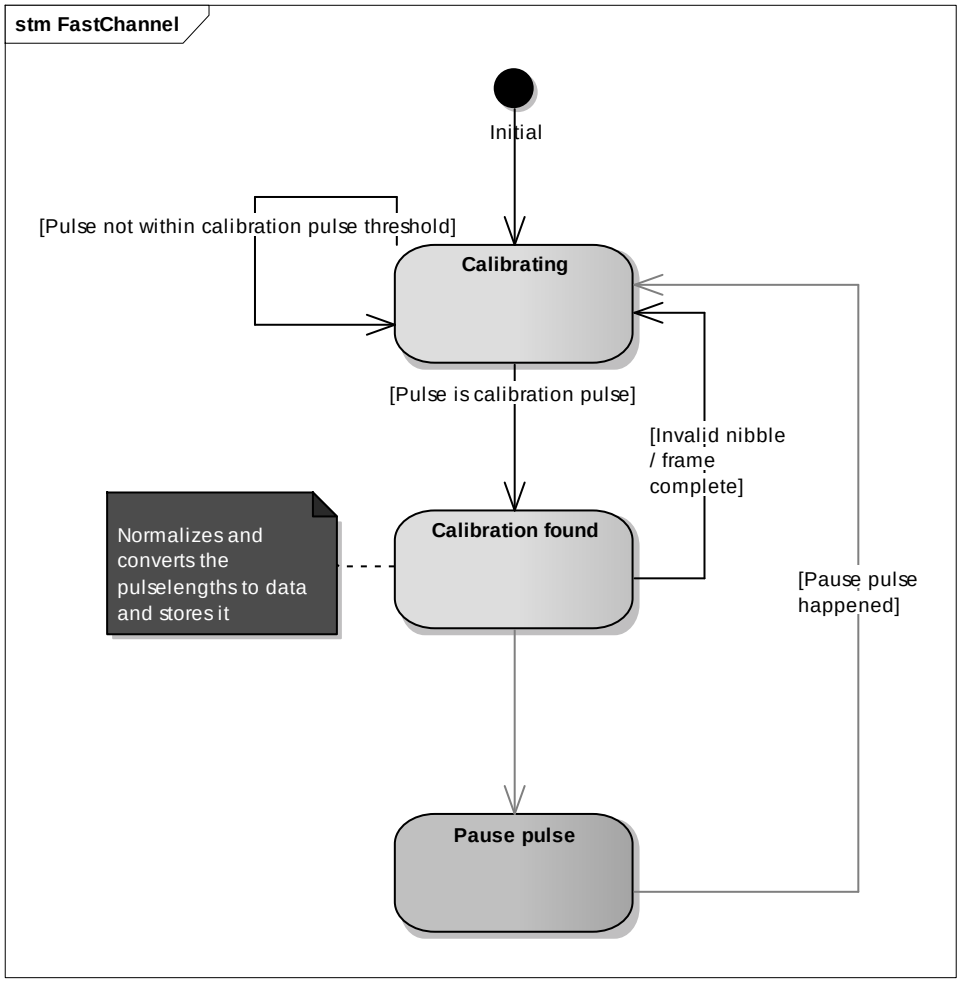


Figure 3-5 Fast channel process

The fast channel process checks incoming pulse lengths, if they are within the configured threshold. Once a calibration pulse is detected, a correction factor is stored for the remaining pulses of the frame and the state is switched.

On successive calls, the pulses are then normalized and converted to actual data values. Once all nibbles of the frame have been received, the 4Bit CRC is checked. On correct CRC checksum, a positive return value is given.

### 3.5.4 Slow channel processing

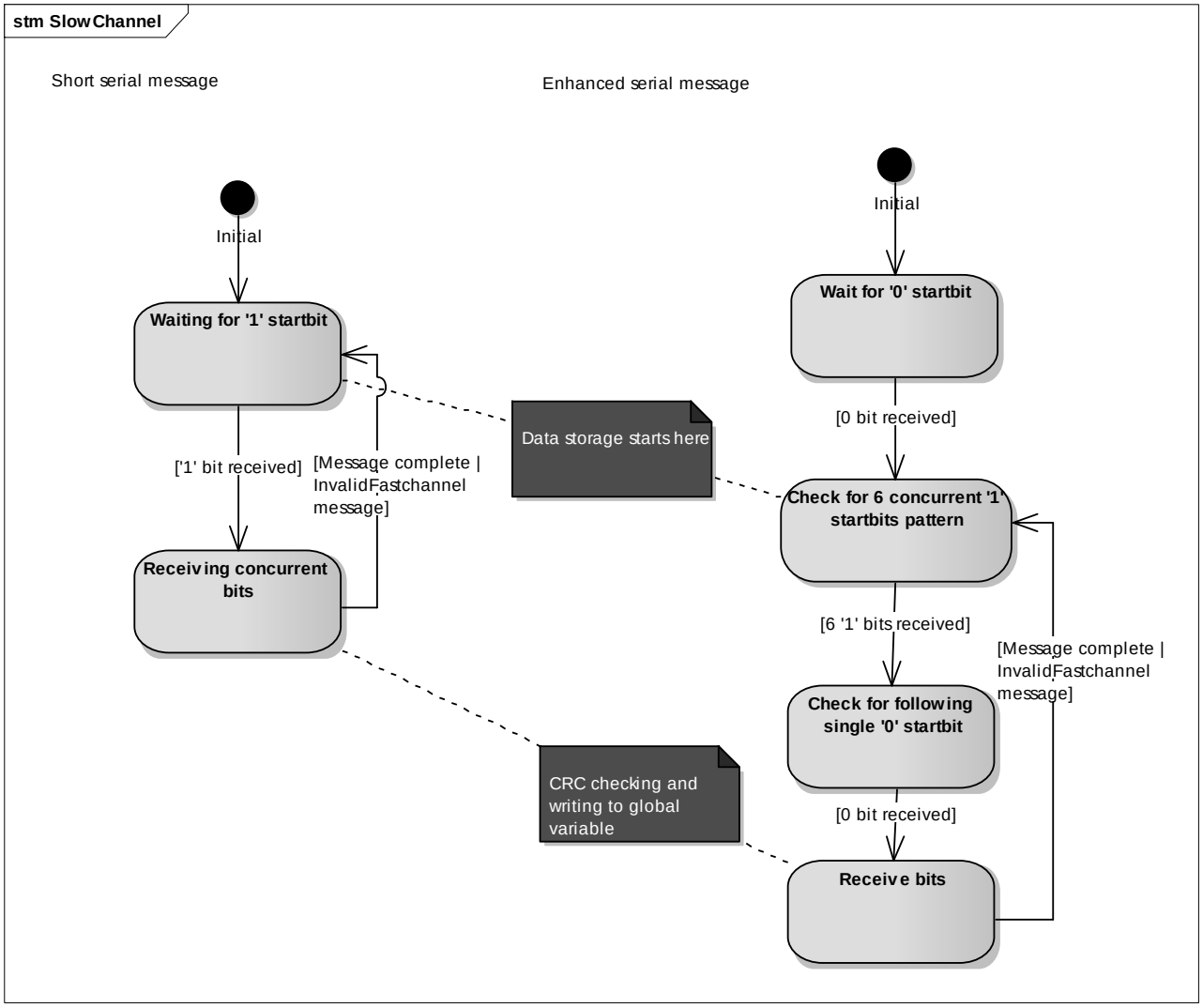


Figure 3-6 Slow channel process

The slow-channel processing will only happen when a fast-channel message has been successfully received and a serial message is configured. The status bits are fed into the state machine and upon a complete message, the data will be written to the global variable.

The state machine and slow channel pulse processing differs depending on whether a Short-Serial-Message or an Enhanced-Serial-Message protocol is used.

## 3.6 Error Handling

### 3.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [5] , if development error reporting is enabled (i.e. pre-compile parameter `SENT_30_ICU_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported Sent_30_Icu ID is 0x8001.

The reported service IDs identify the services which are described in Table 3-2. The following table presents the service IDs and the related services:

| Service ID | Service |
|---|---|
| [0x00] | [Sent_30_Icu_Init] |
| [0x01] | [Sent_30_Icu_DeInit] |
| [0x02] | [Sent_30_Icu_IcuChannelCallback] |
| [0x03] | [Sent_30_Icu_MainFunction] |
| [0x04] | [Sent_30_Icu_GetVersionInfo] |
| [0x05] | [Sent_30_Icu_GetFastChannelData] |
| [0x06] | [Sent_30_Icu_GetSlowChannelData] |

Table 3-1 Service IDs

The errors reported to DET are described in the following table:

| Error Code | Description |
|---|---|
| [0x00] | Invalid pointer: an invalid pointer (NULL_PTR) was given |
| [0x01] | Function was called with uninitialized component |
| [0x02] | The component is already initialized |
| [0x03] | Invalid configuration parameter: Slow channel mode |
| [0x04] | Invalid configuration parameter: Tick time |
| [0x05] | Invalid configuration parameter: CPU tick time |
| [0x06] | Invalid configuration parameter: Data nibbles |

Table 3-2 Errors reported to DET

# 4 Integration

This chapter gives necessary information for the integration of the Sent_30_Icu into an application environment of an ECU.

## 4.1 Scope of Delivery

The delivery of the Sent_30_Icu contains the following static files:

| File Name | Description |
|---|---|
| Sent_30_Icu.c | This is the source file of the component. |
| Sent_30_Icu.h | This is the header file of the component. |
| Sent_30_Icu_Cfg.h | This is the header file of the precompile configuration module. |

Table 4-1 Static files

Modifications should only be made in the precompile configuration module.

## 4.2 Required components

The Sent_30_Icu requires the following MICROSAR components:

| Module | Description |
|---|---|
| ICU | Measures timestamps of falling edges on the SENT SIG line. |
| (DET) | Optional. Reports development errors. |

Table 4-2 Required components

## 4.3 Initialization

To initialize the component in the ECU environment, the following functions have to be called (in this order):

- `Sent_30_Icu_InitMemory(void)`
- `Sent_30_Icu_Init(void)`

## 4.4 Operation

To operate in MainFunction mode, the Sent_30_Icu_MainFunction has to be called periodically. A detailed description of this function can be seen in [3.5.2.1].

**Note**
The Sent_30_E52141 was hardware tested with a cycle time of 10ms.

In IcuCallback mode, the notification function will be automatically called after each measure, if configured correctly.

## 4.5 Deinitialization

To turn the component off, the `Sent_30_Icu_DeInit` function can be called. This stops the ICU timestamping and the state machine.

**Note**
The component can be started again by calling Sent_30_Icu_Init.

# 5 Configuration

In the Sent_30_Icu driver, the attributes can only be configured PRE-COMPILE-TIME.

## 5.1 MICROSAR configuration

The component requires a configuration of the MICROSAR components. The configuration can be done with the Vector DaVinci Configurator tool.

### 5.1.1 ICU

The component needs one IcuChannel with the following attributes:

| Attribute name | Attribute |
|---|---|
| Channel Id | ECU dependent (e.g. 1) |
| Hw Channel | ECU dependent (e.g. EMIOS_0_CH_1) |
| Emios Freeze | false |
| Emios Prescaler | ECU dependent (e.g. EMIOS_PRESCALER_DIVIDE_1) |
| Emios Digital Filter | EMIOS_DIGITAL_FILTER_BYPASSED |
| Emios Bus Select | EMIOS_BUS_INTERNAL_COUNTER |
| Measurement Mode | ICU_MODE_TIMESTAMP |
| Default Start Edge | ICU_FALLING_EDGE |
| Timestamp | ICU_CIRCULAR_BUFFER |
| Timestamp Notification | **NULL_PTR** (MainFunction mode) / **Sent_30_Icu_IcuChannelCallback** (Callback mode) |
| Dma Max. Channel | 1 (recommended) |
| Ext. ISR IFCPR Digital Filter | 0 |
| Unused Wakeup Pins Pull Up | false |
| DMA enable | true (recommended) |
| Disable Ecum Wakeup Notification | false |

Table 5-1 IcuChannel configuration

**Note**

DMA is recommended to enhance the performance of the component. Note that the DMA channel also has to be configured in **Mcu/McuModuleConfiguration/McuDMA/-emioschannel-**

**Without DMA, a complete slow channel frame receive will be very unlikely!**

### 5.1.2 PORT

Port access to the SENT SIG line must be configured:

| Attribute name | Attribute |
|---|---|
| Pin PCR | ECU dependent (e.g. 1) |
| Mode | eMIOS_xxxx (e.g. E0UC1) |
| Level Value | PortPinLevelLow |
| Pin Slew Rate | Slow |
| Direction Changeable | false |

Table 5-2 PORT attributes for the sent input

## 5.2 Configuration with the supplied configuration module

The component ships with a module template to configure the component as easily as possible.

### 5.2.1 Header file

Only the following defines should be modified:

| Sent_30_Icu Configuration Items : Sent_30_Icu_Cfg.h | |
|---|---|
| Configuration Item | Description |
| SENT_30_ICU_DEV_ERROR_DETECT | Activates / deactivates support for DET.<br>*STD_OFF*: The component does not use DET.<br>*STD_ON*: The component uses the DET. |
| SENT_30_ICU_VERSION_INFO_API | Enables / disables the GetVersionInfo API.<br>*STD_OFF*: The component offers the service.<br>*STD_ON*: The component doesn't offer the service. |
| SENT_30_ICU_TIMER_CLOCK_UC_TICKS | Amount of ICU timer ticks per us. |
| SENT_30_ICU_TICK_TIME | SENT protocol tick time (found in sensor specification) |
| SENT_30_ICU_AMOUNT_OF_DATA_NIBBLES | Amount of data nibbles without calibration, CRC and status (found in sensor specification) |
| SENT_30_ICU_PAUSE_PULSE | Defines if a pause pulse is sent (found in sensor specification)<br>*STD_OFF*: No pause pulse<br>*STD_ON*: Pause pulse |
| SENT_30_ICU_SERIAL_FORMAT | Type of serial(slow) message used by the sensor (found in the sensor specification)<br>*SENT_30_ICU_NO_SERIAL_MESSAGE*: No serial message used<br>*SENT_30_ICU_SHORT_SERIAL_MESSAGE*: Short serial message format used.<br>*SENT_30_ICU_ENHANCED_SERIAL_MESSAGE*: Enhanced serial message format used. |

| | |
|---|---|
| SENT_30_ICU_CHANNEL | Configured IcuChannel for timestamping. |
| SENT_30_ICU_USE_MAINFUNCTION | Defines the pulse processing mode.<br>*STD_OFF*: IcuCallback notification is used<br>*STD_ON*: Cyclic mainfunction is used |
| SENT_30_ICU_CALIB_THRESH_PERCENT | Defines the percentage maximum deviation from the calculated tick times. (SENT specification: 20%)<br>*0-100u* |
| SENT_30_ICU_ENABLE_DIAGNOSIS | Defines if the diagnostic information APIs shall be available<br>*STD_OFF*: No diagnostic data collection and API<br>*STD_ON*: Diagnostic data will be collected and APIs are available |
| SENT_30_ICU_USE_NOTIFICATION | Defines if notification functions shall be called upon receiving a complete message<br>*STD_OFF*: No notification call will be made<br>*STD_ON*: Notification will be called upon a successful receive |
| SENT_30_ICU_NOTIFICATION_FUNC_FASTCH | Defines the function to be called upon receiving a complete fast channel message. |
| SENT_30_ICU_NOTIFICATION_FUNC_SLOWCH | Defines the function to be called upon receiving a complete slow channel message. |

Table 5-3 Sent_30_Icu_Cfg.h configuration items

**Practical Procedure**

SENT_30_ICU_TIMER_CLOCK_UC_TICKS can be derived from the MCU settings.

An example (VC54S):

The core clock reference point frequency is 120Mhz

The periphal prescaler is 2

The eMIOS prescaler is 4

120Mhz / 2 / 4 = 15Mhz

15Mhz ^= 15 ticks per us

**Practical Procedure**

While the SENT specification suggests a maximum deviation of 20%, depending on the sensor, it may be required to increase the define

SENT_30_ICU_CALIB_THRESH_PERCENT

or else no calibration pulses will be detected.

# 6 API Description

For an interfaces overview please see Figure 2-3.

## 6.1 Type Definitions

The types defined by the Sent_30_Icu are described in this chapter.

### Sent_30_Icu_FastDataType

This structure is used as memory to store slow channel data when using the short serial message format.

| Struct Element Name | C-Type | Description | Value Range |
|---|---|---|---|
| FastData | uint32 | Memory type to store data from SENT fast channel. (Data[0] = LSB) | 0x00000000 – 0xFFFFFFFF |
| StatusNibble | uint8 | Contains the status nibble of the frame | 0x00 – 0xFF |

Table 6-1 Sent_30_Icu_FastDataType

### Sent_30_Icu_SlowDataType (Short serial message)

This structure is used as memory to store slow channel data when using the short serial message format.

| Struct Element Name | C-Type | Description | Value Range |
|---|---|---|---|
| Data | uint8 | Contains the serial message data. | 0x00 – 0xFF |
| MsgId | uint8 | Contains the serial message identifier. | 0x00 – 0xFF |

Table 6-2 Sent_30_Icu_SlowDataType (Short serial message)

### Sent_30_Icu_SlowDataType (Enhanced serial message)

This structure is used as memory to store slow channel data when using the enhanced serial message format. The actual data sizes depend on a so called configuration bit sent with every by the sensor.

| Struct Element Name | C-Type | Description | Value Range |
|---|---|---|---|
| Data | uint16 | Contains the serial message data. | 0x0000 – 0xFF |
| MsgId | uint8 | Contains the serial message identifier. | 0x00 – 0xFF |

Table 6-3 Sent_30_Icu_SlowDataType (Enhanced serial message)

The following tables describe the provided services of the Sent_30_Icu:

## 6.2 Global variables

| Name | Type | Description |
|---|---|---|
| Sent_30_Icu_FastChannelData | Sent_30_Icu_FastChannelDataType | Contains the fast channel data from the SENT sensor. |
| Sent_30_Icu_SlowChannelData | Sent_30_Icu_SlowChannelDataType | Contains the slow channel data from the SENT sensor. (Only if serial message is configured) |

Table 6-4 Sent_30_Icu global variables

## 6.3 Sent_30_Icu_Init

| Prototype | |
|---|---|
| `void Sent_30_Icu_Init(void)` | |
| **Parameter** | |
| – | - |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Performs the initialization described in [3.5.1]. It calculates thresholds to save time in future function calls and configures the ICU to start timestamping. It then sets the module status to initialized. | |
| **Particularities and Limitations** | |
| > The component must be uninitialized | |

| Call context |
| --- |
| > System initialization |

Table 6-5 Sent_30_Icu_Init

## 6.4 Sent_30_Icu_DeInit

| Prototype | |
| --- | --- |
| `void Sent_30_Icu_DeInit(void)` | |
| **Parameter** | |
| – | - |
| **Return code** | |
| – | - |
| **Functional Description** | |
| This function deinitializes the module Sent_30_Icu. It stops the ICU timestamping and the state machine. | |
| **Particularities and Limitations** | |
| > The module must be initialized | |
| Call context | |
| > System shutdown / resource management | |

Table 6-6 Sent_30_Icu_DeInit

## 6.5 Sent_30_Icu_IcuChannelCallback

| Prototype | |
| --- | --- |
| `void Sent_30_Icu_IcuChannelCallback(void)` | |
| **Parameter** | |
| – | - |
| **Return code** | |
| – | - |
| **Functional Description** | |
| The callback function creates a delta from the last and current ICU timestamp and processes the pulse length. More details in [3.5.2.2]. | |
| **Particularities and Limitations** | |
| > The module must be initialized<br>> The module must be configured in callback-mode | |
| Call context | |
| > On every falling edge on SENT_IN by the ICU | |

Table 6-7 Sent_30_Icu_IcuChannelCallback

## 6.6 Sent_30_Icu_MainFunction

| Prototype | |
|---|---|
| `void` **`Sent_30_Icu_MainFunction`**`(void)` | |
| **Parameter** | |
| – | - |
| **Return code** | |
| – | - |
| **Functional Description** | |
| This function iterates over all new timestamps since the last call and processes them.<br>More details in [3.5.2.1]. | |
| **Particularities and Limitations** | |
| > The module must be initialized | |
| Call context | |
| > Cyclic (e.g. 10ms) | |

Table 6-8 Sent_30_Icu_MainFunction

## 6.7 Sent_30_Icu_GetVersionInfo

| Prototype | |
|---|---|
| `void` **`Sent_30_Icu_GetVersionInfo`**`(Std_VersionInfoType* versioninfo)` | |
| **Parameter** | |
| `versioninfo` | Pointer to type to store the module version info. |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Requests module version information. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > Application | |

Table 6-9 Sent_30_Icu_GetVersionInfo

## 6.8 Sent_30_Icu_GetFastChannelData

| Prototype |
|---|
| `void` **`Sent_30_Icu_GetFastChannelData`** `(Sent_30_Icu_FastDataType* DataPtr)` |

| Parameter | |
|---|---|
| DataPtr | Location where to store the data |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Sent_30_Icu_GetFastChannelData writes the last successfully received fast channel data (raw data + status nibble) of the SENT channel to the pointer location. | |
| **Particularities and Limitations** | |
| > The module must be initialized | |
| Call context | |
| > Application | |

Table 6-10 Sent_30_Icu_GetFastChannelData

## 6.9 Sent_30_Icu_GetSlowChannelData

| Prototype | |
|---|---|
| `void Sent_30_Icu_GetSlowChannelData (Sent_30_Icu_SlowDataType* DataPtr)` | |
| **Parameter** | |
| DataPtr | Location where to store the data |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Sent_30_Icu_GetSlowChannelData writes the last successfully received slow serial channel data of the SENT channel to the pointer location. | |
| **Particularities and Limitations** | |
| > The module must be initialized | |
| > Call context | |
| > Application | |

Table 6-11 Sent_30_Icu_GetSlowChannelData

## 6.10 Sent_30_Icu_GetDiagnosticValues

| Prototype | |
|---|---|
| `void Sent_30_Icu_GetDiagnosticValues (Sent_30_Icu_DiagDataType* DataPtr)` | |
| **Parameter** | |
| DataPtr | Location where to store the data |
| **Return code** | |
| – | - |

| Functional Description |
| --- |
| The function writes the current collected diagnostic data values to the given location. |
| **Particularities and Limitations** |
| > The module must be initialized |
| Call context |
| > Application |

Table 6-12 Sent_30_Icu_GetDiagnosticValues

## 6.11 Sent_30_Icu_ResetDiagnosticValues

| Prototype | |
| --- | --- |
| `void` **`Sent_30_Icu_ResetDiagnosticValues`** `(void)` | |
| **Parameter** | |
| – | - |
| **Return code** | |
| – | - |
| **Functional Description** | |
| The function resets the diagnostic counter values to 0. | |
| **Particularities and Limitations** | |
| > The module must be initialized | |
| Call context | |
| > Application | |

Table 6-13 Sent_30_Icu_ResetDiagnosticValues

# 7 Glossary and Abbreviations

## 7.1 Glossary

| Term | Description |
|---|---|
| | |

Table 7-1 Glossary

## 7.2 Abbreviations

| Abbreviation | Description |
|---|---|
| API | Application Programming Interface |
| AUTOSAR | Automotive Open System Architecture |
| BSW | Basis Software |
| DET | Development Error Tracer |
| ECU | Electronic Control Unit |
| MICROSAR | Microcontroller Open System Architecture (the Vector AUTOSAR solution) |
| SWC | Software Component |
| SWS | Software Specification |
| ICU | Input capture unit |
| SENT | Single Edge Nibble Transmission |

Table 7-2 Abbreviations

# 8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com