UART driver user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This user guide is intended to enable users to integrate the universal asynchronous receiver/transmitter (UART) driver software for the TRAVEO™ T2G family MCUs. Furthermore, this user guide is intended to enable users to integrate the video UART (VUART) driver software as Add-On feature for the UART driver.

This document describes responsibilities of the integrator in charge of integrating the UART and VUART drivers with the basic software (BSW) stack. This document also provides detailed information on safety, configuration, and functions along with examples of usage of significant features.

Intended audience

This document is intended for anyone who uses the UART driver and VUART driver of the TRAVEO™ T2G family.

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus, and submenus
Italics	Denotes file/folder names
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, command line inputs, and code snippets

Abbreviations and definitions

Table 2 Abbreviations

Abbreviation	Definition
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASIL	Automotive Safety Integrity Level
AUTOSAR	AUTomotive Open System Architecture
BSW	Basic Software
CAN	Controller Area Network
CDD	Complex Device Driver or Complex Driver
DEM	Diagnostics Event Manager (MCAL module)
DET	Development Error Tracer
DMA	Direct Memory Access
DW	Data Wire

About this document

Abbreviation	Definition
EB tresos Studio	Elektrobit Automotive configuration framework
GCE	Generic Configuration Editor
I2C	Inter-Integrated Circuit
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
MCAL	Microcontroller Abstraction Layer
PCLK	Peripheral Clock
SCB	Serial Communication Block
SchM	Scheduler Manager
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
VUART	Video UART

Related documents

This user manual should be read in conjunction with the following documents:

AUTOSAR requirements and specifications

[1] General specification of basic software modules, AUTOSAR release 4.2.2

[2] Complex driver design and integration guideline, AUTOSAR release 4.2.2

[3] Specification of standard types, AUTOSAR release 4.2.2

[4] Specification of default error tracer, AUTOSAR release 4.2.2

Elektrobit automotive documentation

[5] EB tresos Studio for ACG8 user's guide

Hardware documentation

The hardware documents are listed in the delivery notes.

Related standards and norms

[6] Layered software architecture, AUTOSAR release 4.2.2

Table of contents

About this document 1
Table of contents 3
1 General overview 8
1.1 Introduction to the UART driver 8
1.2 User profile 8
1.3 Embedding in the AUTOSAR environment 9
1.4 Supported hardware 9
1.5 Development environment 10
1.6 Character set and encoding 10
1.7 How to read this document 10
2 UART driver 12
2.1 Using the UART driver 12
2.1.1 Installation and prerequisites 12
2.1.2 Configuring the UART driver 12
2.1.2.1 Configuration outline 12
2.1.3 Adapting your application 14
2.1.4 Start the build process 16
2.1.5 Measuring the stack consumption 16
2.1.6 Memory mapping 16
2.1.6.1 Memory allocation keywords 16
2.1.6.2 Restriction for memory allocation 17
2.2 Structure and dependencies 17
2.2.1 Static files 17
2.2.2 Configuration files 18
2.2.3 Generated files 18
2.2.4 Dependencies 18
2.2.4.1 PORT driver 18
2.2.4.2 MCU driver 18
2.2.4.3 AUTOSAR OS 19
2.2.4.4 BSW scheduler 19
2.2.4.5 DET 19
2.2.4.6 DEM 19
2.2.4.7 Error callout handler 19
2.3 EB tresos Studio configuration interface 19
2.3.1 General configuration 19
2.3.2 UART configuration 21
2.3.3 UART communication configuration 22
2.3.4 UART DW configuration 24
2.3.5 Other modules 25
2.3.5.1 PORT driver 25
2.3.5.2 MCU driver 25
2.3.5.3 DET 25
2.3.5.4 DEM 25
2.3.5.5 AUTOSAR OS 25
2.3.5.6 BSW scheduler 25
2.3.5.7 Software for DMA 25
2.4 Functional description 26

Table of contents

2.4.1 Overview of UART driver functionality 26
2.4.1.1 Data reception overview 26
2.4.1.2 State transition 27
2.4.2 UART driver functionality and operation 29
2.4.2.1 Initialize the UART driver 29
2.4.2.2 Prepare the external buffer for reception and start auto reception 29
2.4.2.3 Start synchronous transmit 30
2.4.2.4 Start asynchronous transmit 31
2.4.2.5 Start synchronous receive 31
2.4.2.6 Start asynchronous receive 32
2.4.2.7 Stop / termination and restart 33
2.4.2.8 Get driver information 35
2.4.2.9 Change the communication setting 36
2.4.2.10 Deinitialize the driver 37
2.4.3 What should be included 37
2.4.4 System initialization 37
2.4.5 Runtime reconfiguration 37
2.4.6 API parameter checking 37
2.4.6.1 Vendor-specific development errors 38
2.4.7 Production errors 38
2.4.7.1 Recoverable failure 38
2.4.7.2 Unrecoverable failure 38
2.4.8 Reentrancy 39
2.4.9 Sleep mode 39
2.4.10 Debugging support 39
2.4.11 Execution time dependencies 39
2.4.12 Deviation from AUTOSAR 40
2.5 Related hardware resources 40
2.5.1 Ports and pins 40
2.5.2 Timer 40
2.5.3 Interrupts 40
2.5.4 DMA 41
3 VUART driver 42
3.1 Using the VUART driver 42
3.1.1 Installation and prerequisites 42
3.1.2 Configuring the VUART driver 42
3.1.2.1 Configuration outline 42
3.1.3 Adapting your application 42
3.1.4 Start the build process 44
3.1.5 Measuring the stack consumption 44
3.1.6 Memory mapping 44
3.1.6.1 Memory allocation keywords 44
3.1.6.2 Restriction for memory allocation for data buffer 44
3.2 Structure and dependencies 45
3.2.1 Static files 45
3.2.2 Configuration files 45
3.2.3 Generated files 45
3.2.4 Dependencies 46
3.2.4.1 UART driver 46
3.2.4.2 DET 46

Table of contents

3.2.4.3 Error callout handler ... 46
3.3 EB tresos Studio configuration interface ... 46
3.3.1 General configuration ... 46
3.3.2 VUART configuration ... 47
3.3.3 Other modules ... 48
3.3.3.1 UART driver ... 48
3.3.3.2 PORT driver ... 48
3.3.3.3 DET ... 48
3.4 Functional description ... 49
3.4.1 Overview of VUART driver functionality ... 49
3.4.1.1 Double buffering mechanism ... 49
3.4.1.2 Frame data structure ... 50
3.4.1.3 State transition ... 51
3.4.1.4 Transmission overview ... 52
3.4.2 VUART driver functionality and operation ... 54
3.4.2.1 Start Vuart ... 54
3.4.2.2 Get driver information ... 55
3.4.2.3 Stop / termination and restart ... 56
3.4.3 What should be included ... 56
3.4.4 System initialization ... 56
3.4.5 Runtime reconfiguration ... 57
3.4.6 API parameter checking ... 57
3.4.6.1 Vendor-specific development errors ... 57
3.4.7 Production errors ... 57
3.4.8 Reentrancy ... 57
3.4.9 Sleep mode ... 57
3.4.10 Debugging support ... 58
3.4.11 Execution time dependencies ... 58
3.4.12 Deviation from AUTOSAR ... 58
3.5 Related hardware resources ... 58
3.5.1 Ports and pins ... 58
3.5.2 Timer ... 58
3.5.3 Interrupts ... 58
4 Appendix A: UART API and external definitions ... 59
4.1 Include files ... 59
4.2 Data types ... 59
4.2.1 Uart_ChannelIdType ... 59
4.2.2 Uart_BufferType ... 59
4.2.3 Uart_BufferSizeType ... 59
4.2.4 Uart_CommIdType ... 59
4.2.5 Uart_ChannelTxStatusType ... 60
4.2.6 Uart_ChannelRxStatusType ... 60
4.2.7 Uart_ConfigType ... 61
4.2.8 Uart_HeaderType ... 61
4.3 Constants ... 61
4.3.1 Error codes ... 61
4.3.2 Version / module information ... 62
4.3.3 API service ID ... 62
4.4 Functions ... 63
4.4.1 Uart_Init ... 63

Table of contents

4.4.2 Uart_DeInit 64
4.4.3 Uart_SetCommMode 64
4.4.4 Uart_GetCommMode 66
4.4.5 Uart_GetTxStatus 67
4.4.6 Uart_GetRxStatus 68
4.4.7 Uart_SetupEb 69
4.4.8 Uart_SyncTransmit 70
4.4.9 Uart_SyncReceive 71
4.4.10 Uart_AsyncTransmit 72
4.4.11 Uart_AsyncReceive 73
4.4.12 Uart_CancelTransmit 74
4.4.13 Uart_CancelReceive 75
4.4.14 Uart_StartUartRx 76
4.4.15 Uart_GetRingBufInfo 77
4.4.16 Uart_GetVersionInfo 78
4.4.17 Uart_GetTxBufferInfo 79
4.4.18 Uart_GetTxIntStatus 80
4.4.19 Uart_AsyncTransmitWithHeader 81
4.5 Scheduled functions 81
4.6 Interrupt service routine 82
4.6.1 Uart_Interrupt_SCB<n>_CatX 82
4.7 Required callback functions 82
4.7.1 UART notification function 82
4.7.1.1 Uart_TxNotification 83
4.7.1.2 Uart_RxNotification 83
4.7.1.3 Uart_TxErrorNotification 83
4.7.1.4 Uart_RxErrorNotification 84
4.7.2 DET 84
4.7.2.1 Det_ReportError 84
4.7.3 DEM 85
4.7.3.1 Dem_ReportErrorStatus 85
4.7.4 Error Callout functions 85
4.7.4.1 Error Callout function 85
5 Appendix B: VUART API and external definitions 86
5.1 Include files 86
5.2 Data types 86
5.2.1 Vuart_ChannelIdType 86
5.2.2 Vuart_BufferType 86
5.2.3 Vuart_HeaderType 86
5.3 Constants 86
5.3.1 Error codes 86
5.3.2 Version / module information 87
5.3.3 API service ID 87
5.4 Functions 88
5.4.1 Vuart_GetVersionInfo 88
5.4.2 Vuart_Start 89
5.4.3 Vuart_Stop 90
5.4.4 Vuart_UpdateBuffer 91
5.4.5 Vuart_GetCurrentRow 92
5.5 Scheduled functions 93

Table of contents

5.5.1 Vuart_MainFunction_Ch[n] 93
5.6 Interrupt service routine 93
5.7 Required callback functions 94
5.7.1 VUART notification function 94
5.7.1.1 Vuart_SwapCompleted 94
5.7.2 DET 95
5.7.2.1 Det_ReportError 95
5.7.3 DEM 95
5.7.4 Error Callout functions 96
6 Appendix C: Registers 97
6.1 Access register table 97
Revision history 103

1 General overview

1.1 Introduction to the UART driver

The UART driver is a complex driver, which enables you to support UART communication on special output pins of the MCU.

The UART driver provides services for reading from and writing to devices connected via UART buses. The UART driver provides access to UART communication for multiple peripherals. Only the standard UART mode is supported.

The VUART driver is a complex driver, which enables you to support VUART communication on special output pins of the MCU.

The VUART driver provides services for transmitting image pixel data to light-source devices via UART buses by using the UART driver.

The UART and VUART drivers are not responsible for initializing or configuring hardware ports. This is done by the PORT driver.

The UART and VUART drivers conform to the AUTOSAR standard and are implemented according to the AUTOSAR complex driver design and integration guideline.

1.2 User profile

This guide is intended for users with a basic knowledge of the following domains:

- Embedded systems
- C programming language
- AUTOSAR standard
- Target hardware architecture

1.3 Embedding in the AUTOSAR environment

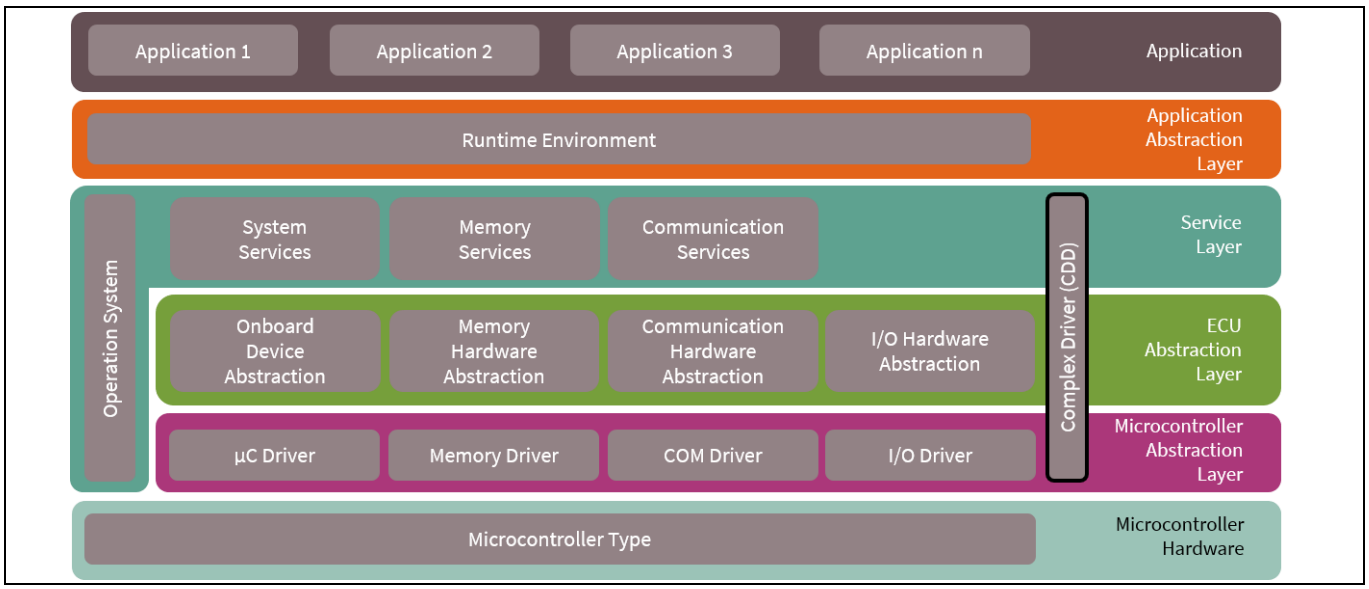

Figure 1 Overview of AUTOSAR software layers

Figure 1 depicts the layered AUTOSAR software architecture. There can be multiple complex device drivers (CDD), UART and VUART drivers (**Figure 2**) among them. The UART and VUART drivers have similar functionality as the microcontroller abstraction layer (MCAL).

For an exact overview of the AUTOSAR layered software architecture, see layered software architecture**[6]**.

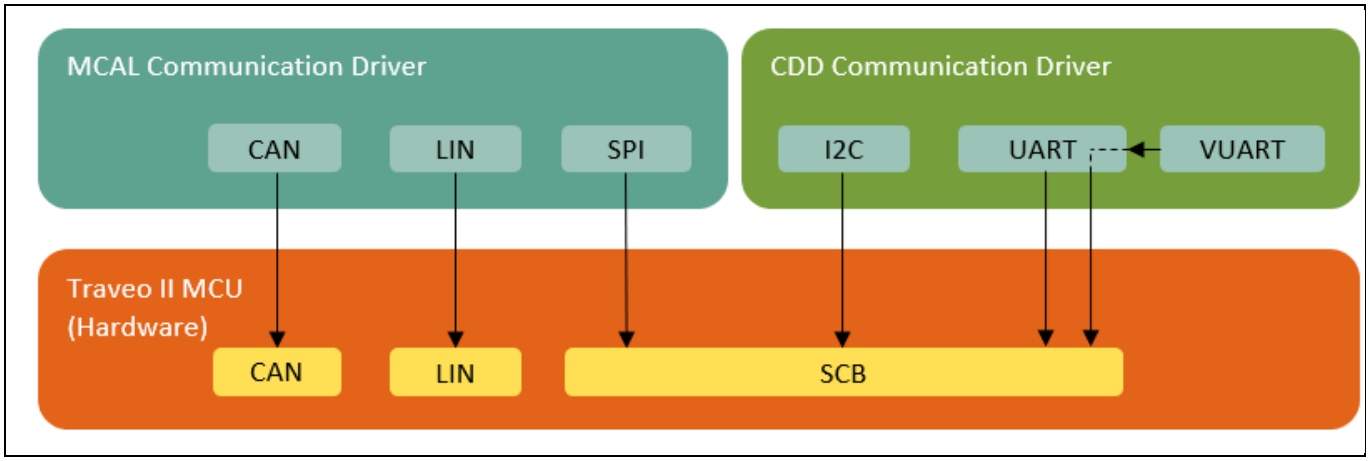

Figure 2 UART and VUART drivers as a complex driver

1.4 Supported hardware

This version of the UART and VUART drivers support the TRAVEO™ T2G family of microcontrollers. No special external hardware devices are required.

The supported derivatives are listed in the release notes.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2. The Base, Make, Mcu, Port, and Resource modules are needed for the proper functionality of the UART and VUART drivers.

1.6 Character set and encoding

All source code files of the UART and VUART drivers are restricted to the ASCII character set. The files are encoded in UTF-8 format, with only the 7-bit subset (values 0x00 … 0x7F) being used.

1.7 How to read this document

This document describes both the UART and VUART drivers. However, the combination of these drivers depends on the use case. At first, you should confirm the use case, and then read the corresponding chapter accordingly.

Table 3 Use case and how to read this document

Use case	How to read this document
Only use the UART driver	You should read chapter **2** and chapter **4**, chapter **6**.
Only use the VUART driver	You should read chapter **3** and chapter **5**, chapter **6**.
Both the UART and VUART drivers in different SCB channel	You should read all chapter to construct the entire environment. For UART channel, you should apply chapter **2** and related chapter. For VUART channel, you should apply chapter **3** and related chapter.

Table 4 shows the corresponding chapter for each use case.

Table 4 Corresponding chapter for each use case

Item	UART only	VUART only	Both UART and VUART
Installation	See **2.1.1**	See **3.1.1**	See **3.1.1**
Configuring driver	See **2.1.2**	See **2.1.2** and **3.1.2**	See **2.1.2** and **3.1.2**
Adapting your application	See **2.1.3**	See **3.1.3**	See **2.1.3** and **3.1.3**
Build process	See **2.1.4**	See **3.1.4**	See **3.1.4**
Measuring the stack consumption	See **2.1.5**	See **3.1.5**	See **3.1.5**
Memory mapping	See **2.1.6**	See **2.1.6** and **3.1.6**	See **2.1.6** and **3.1.6**
Structure (Files)	See **2.2** (excepts Dependencies)	See **2.2** and **3.2** (excepts Dependencies)	See **2.2** and **3.2** (excepts Dependencies)
Dependencies	See **2.2.4**	See **2.2.4** and **3.2.4**	See **2.2.4** and **3.2.4**
Configuration interface	See **2.3** (excepts other modules)	See **2.3** and **3.3** (excepts other modules)	See **2.3** and **3.3** (excepts other modules)
Configuration interface (other modules)	See **2.3.5**	See **2.3.5** and **3.3.3**	See **2.3.5** and **3.3.3**
Functional description	See **2.4**	See **3.4**	See **2.4** and **3.4**

Item	UART only	VUART only	Both UART and VUART
Related hardware resources	See **2.5**	See **3.5**	See **2.5** and **3.5**
Appendix	See **4** and **6**	See **5** and **6**	See **4** and **5** and **6**

2 UART driver

This chapter describes UART specific information.

2.1 Using the UART driver

This chapter describes all necessary steps to incorporate the UART driver into your application.

This chapter also contains the necessary information to incorporate the VUART driver. If you want to incorporate the VUART driver, see **3 VUART driver** at first.

2.1.1 Installation and prerequisites

Note: *Before continuing with this chapter, see the* EB tresos Studio for ACG8 user's guide **[5]**. *You can find the required basic information about the installation procedure of EB tresos AUTOSAR components and the use of EB tresos and the EB tresos AUTOSAR build environment. You will also find information on how to set up and integrate your own application within the EB tresos AUTOSAR build environment there.*

The installation of the UART driver corresponds with the general installation procedure for EB tresos AUTOSAR components given in the documents mentioned above.

This document assumes that you have set up your project using the application template. This template provides the necessary folder structure, project, and Makefiles needed to configure and compile your application within the build environment. You must be familiar with the use of the command shell.

2.1.2 Configuring the UART driver

The UART driver can be configured with any AUTOSAR-compliant GCE tool. Save the configuration in a separate file, for example, *Uart.epc*. For more information about the UART driver configuration, see **2.3 EB tresos Studio configuration interface.**

2.1.2.1 Configuration outline

Table 5 shows the outline of the UART driver configuration.

Table 5 Configuration container and parameters

Container / parameter	Description
`UartDemEventParameterRefs`	Turns the diagnostics event manager (DEM) feature used in the UART driver ON/OFF.
`UART_DEM_RECOVERABLE_FAILURE`	Specifies the DEM event for recoverable failures.
`UART_DEM_UNRECOVERABLE_FAILURE`	Specifies the DEM event for unrecoverable failures.
`UartGeneral`	Container for parameters that turn the optional APIs and features ON/OFF.
`UartVersionInfoApi`	Specifies whether the version info API is used.
`UartErrorCalloutFunction`	Specifies the name of the error callout function.
`UartTxNotificationCalloutFunction`	Specifies the name of the Tx notification function.
`UartRxNotificationCalloutFunction`	Specifies the name of the Rx notification function.
`UartTxErrorNotificationCalloutFunction`	Specifies the name of the Tx error notification function.

Container / parameter	Description
UartRxErrorNotificationCalloutFunction	Specifies the name of the Rx error notification function.
UartDevErrorDetect	Specifies whether development error detection is used.
UartOsCounterRef	Specifies the OS counter which is used by the UART driver.
UartConfigSet	Container for setting the whole UART channels configuration.
UartChannelConfig	Container for setting individual channel configuration.
UartChannelId	Specifies the ID used to access a UART channel.
UartScbChannelNumber	Specifies the SCB (hardware) resource used for a UART channel.
UartDefaultCommConfigID	Specifies the default communication setting ID.
UartMaxBuffersize	Specifies the maximum ring buffer size for UART reception. (Not available when VUART is enabled for channel)
UartRxBlockThreshold	Specifies the threshold for Rx notification in number of bytes. (Not available when VUART is enabled for channel)
UartScbCommConfig	Container for communication-related settings.
UartCommConfigID	Specifies the ID for communication settings.
UartClockRef	Specifies the SCB input clock reference point in the MCU configuration.
UartClockRefInfo	Specifies the SCB input clock speed (Hz).
UartBreakLevel	Specifies the break detection level. (Not available when VUART is enabled for channel)
UartBreakwidth	Specifies the length in bits of a break detection interval. (Not available when VUART is enabled for channel)
UartTxMsbFirst	Specifies the first transfer bit for Tx communication.
UartRxMsbFirst	Specifies the first receive bit for Rx communication. (Not available when VUART is enabled for channel)
UartTxParity	Specifies the parity bit for Tx communication.
UartRxParity	Specifies the parity bit for Rx communication. (Not available when VUART is enabled for channel)
UartOpenDrainSelector	Specifies the output mode to either normal operation mode or open-drain operation mode.
UartFrameErrorDataDrop	Specifies the behavior when a frame error is detected. (Not available when VUART is enabled for channel)
UartParityErrorDataDrop	Specifies the behavior when a parity error is detected. (Not available when VUART is enabled for channel)
UartTxTriggerLevel	Specifies the trigger level for Tx operation.
UartTxStopbitWidth	Specifies the stop period width for Tx communication.
UartRxStopbitWidth	Specifies the stop period width for Rx communication. (Not available when VUART is enabled for channel)
UartTxDataWidth	Specifies the data bit count for Tx communication.

Container / parameter	Description
UartRxDataWidth	Specifies the data bit count for Rx communication. (Not available when VUART is enabled for channel)
UartMedianFilterEnable	Specifies the median filter usage. (Not available when VUART is enabled for channel)
UartOversamplingfactor	Specifies the oversampling factor for UART communication.
UartCalculatedBaudRate	Specifies the output/input baud rate.
UartTxTimeout	Specifies the max execution period for synchronous Tx operation. (Not available when VUART is enabled for channel)
UartRxTimeout	Specifies the max execution period for synchronous Rx operation. (Not available when VUART is enabled for channel)
UartIncludeFile	Specifies the user-defined include file.
UartDWConfig	Container for setting the DW channel configurations. (Available when VUART is enabled for channel)
UartDWChannel	Specifies the DW channel (hardware) resource for UART communication. (Available when VUART is enabled for channel)
UartDWCRCMode	Specifies the CRC mode as NO CRC, CRC 16 bit and CRC 32 bit. (Available when VUART is enabled for channel)
UartDWCRCPolynomial	Specifies the CRC Polynomial value. (Available when VUART is enabled for channel)
UartDWCRCSeed	Specifies the CRC Seed value. (Available when VUART is enabled for channel)
UartDWCRCIncludeHeader	Specifies whether the row header is included in the CRC calculation. (Available when VUART is enabled for channel)

2.1.3 Adapting your application

To use the UART driver in your application, include the header files of the UART, MCU, and PORT drivers by adding the following lines of code to your source code file.

Code Listing 1 Example for include

```
001     #include "Mcu.h"    /* AUTOSAR MCU Driver */
002     #include "Port.h"   /* AUTOSAR PORT Driver */
003     #include "Uart.h"   /* UART Driver */
```

This makes all required functions, data types, and symbolic names known to the application.

To use the UART driver, you must configure appropriate port pins, SCB clock settings, and UART interrupts in the PORT driver, MCU driver, and OS. For detailed information, see **2.5 Related hardware resources**.

You must initialize the MCU, PORT, and UART driver in the following order:

Code Listing 2 Example for initialize sequence

```
001         Mcu_Init(&Mcu_Config[0]);
002         Port_Init(&Port_Config[0]);
003         Uart_Init(&Uart_Config[0]);
```

The function `Port_Init()` is called with a pointer to a structure of type `Port_ConfigType`, which is exported by the PORT driver itself.

The interrupt service routine must be configured in the AUTOSAR OS for each UART peripheral, as described in **2.5.3 Interrupts**.

Note: If the UART channel is used for VUART, then corresponding SCB channel is not required to configured in AUTOSAR OS. Because the VUART is not used the interrupt service routine.

All required input clocks for the configured hardware units (SCB) must be activated before initializing the UART driver. See **2.2.4.2 MCU driver**.

Your application must provide the notification functions that you configured and their declarations. The file containing the declarations must be included using the `UartIncludeFile` parameter. See the following example of a function declaration and definition for the notification functions:

Code Listing 3 Example for declaration

```
001         extern void My_TxNotification(void);
002         extern void My_RxNotification(void);
```

Code Listing 4 Example for definition

```
001         void My_TxNotification(void)
002         {
003           /* Insert your code here */
004         }
005         void My_RxNotification(void)
006         {
007           /* Insert your code here */
008         }
```

These notification functions are called from an interrupt context.

Note: If the UART channel is used for VUART, this notification is not used but need to implement as described. Even though the interrupt service routine is not used for VUART.

2.1.4 Start the build process

Do the following to build your application.

Note: *For a clean build, use the build command with target* `make clean_all` *before the following.*

1. In the command shell, type the following command. This will generate the necessary configuration-dependent files. See **2.2.3 Generated files**.

   ```
   > make generate
   ```

2. Type the following command to generate file dependency lists:

   ```
   > make depend
   ```

3. Compile and link the application with the following command:

   ```
   > make (optional target: all)
   ```

The application is built now. All files are compiled and linked to a binary file which can be downloaded to the target hardware.

2.1.5 Measuring the stack consumption

Do the following to measure the stack consumption. The Base module is needed for a proper measurement.

Note: *All files (including library files) should be rebuilt with the* 'stack analysis' *compiler option. The executable file built in this step must be used for stack consumption measurement only.*

1. Add the following compiler option to the Makefile to enable stack consumption measurement:
   `-DSTACK_ANALYSIS_ENABLE`
2. Type the following command to clean library files:
   `make clean_lib`
3. Follow the build process described in **2.1.4 Start the build process**.
4. Follow the instructions in the release notes and measure the stack consumption.

2.1.6 Memory mapping

The *Uart_MemMap.h* file in the *$(TRESOS_BASE)/plugins/Uart_MemmapSample* directory is a sample. This file is replaced by the file generated by the MEMMAP module. The input to the MEMMAP module is described in the *Uart_Bswmd.arxml* file in the *$(PROJECT_ROOT)/output/generate_swcd/swcd* directory of your project folder.

2.1.6.1 Memory allocation keywords

- `UART_START_SEC_CODE_ASIL_B / UART_STOP_SEC_CODE_ASIL_B`

  This memory section type is CODE. All executable code is allocated in this section.

- `UART_START_SEC_CONST_ASIL_B_UNSPECIFIED / UART_STOP_SEC_CONST_ASIL_B_UNSPECIFIED`

  This memory section type is CONST. All configuration data is allocated in this section.

- `UART_START_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED / UART_STOP_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED`

  This memory section type is VAR. All non-initialized variables without alignment restrictions are allocated in this section.

- `UART_START_SEC_VAR_INIT_ASIL_B_UNSPECIFIED / UART_STOP_SEC_VAR_INIT_ASIL_B_UNSPECIFIED`

  This memory section type is VAR. All initialized variables without alignment restrictions are allocated in this section.

2.1.6.2 Restriction for memory allocation

At least one of the UART channel is used for VUART, the following restriction should be applied.
The CPU has a private cache that is not shared with the DMA bus master. Therefore, you must ensure that the data accessed by DMA are in uncached memory regions. The UART driver does not support the memory allocation of DMA-related memory and data buffer to the CPU's tightly coupled memories (TCMs) and internal video RAM (VRAM).

- The section surrounded by UART_START_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED / UART_STOP_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED: Because the UART is using DMA for data transactions, this section must be allocated to a user-specific memory region configured by the MPU as write-through or non-cacheable.

Note: *These restrictions are applied only to the Cortex®-M7 CPU because it includes TCMs, VRAM and cache. These restrictions do not apply when using the Cortex®-M4 CPU. The areas mentioned above must be accessible through DMA and require 4-byte alignment.*

2.2 Structure and dependencies

The UART driver consists of static, configuration, and generated files.

2.2.1 Static files

Table 6 Static files

Folder	Description
$(PLUGIN_PATH)=$(TRESOS_BASE)/plugins/Uart_TS_*	Path to the UART driver plugin.
$(PLUGIN_PATH)/lib_src	Contains all static source files of the UART driver. These files contain the functionality of the driver that does not depend on the current configuration. The files are used to build a static library.
$(PLUGIN_PATH)/src	Comprises configuration-dependent source files or derivative-specific files. Each file will be rebuilt when the configuration is changed. All necessary source files will automatically be compiled and linked during the build process, and all include paths will be set if the UART driver is enabled.
$(PLUGIN_PATH)/include	Basic public include directory needed by the user to include *Uart.h*.
$(PLUGIN_PATH)/autosar	Contains the AUTOSAR ECU parameter definition with vendor-, architecture-, and derivative-specific adaptations to create a correctly matching parameter configuration for the UART driver.

2.2.2 Configuration files

The configuration of the UART driver is done via EB tresos Studio. The file containing the UART driver's configuration is named *Uart.xdm* and is in the *$(PROJECT_ROOT)/config* directory. This file serves as the input to generate configuration-dependent source and header files during the build process.

2.2.3 Generated files

During the build process, the following files are generated based on the current configuration description. They are in the *output/generated* subfolder of your project folder.

Table 7 Generated files

Files	Description
include/Uart_Cfg.h	Contains all symbolic names for the configured UART channels.
include/Uart_PBcfg.h	Contains the configured constants for the UART driver.
include/Uart_ExternalInclude.h	Contains the include directives for user-configured external header files.
include/Uart_Irq.h	Contains the declaration of ISR functions.
src/Uart_PBcfg.c	Contains the configured constants for the UART driver.
src/Uart_Irq.c	Contains the definition of ISR functions.
swcd/Uart_Bswmd.arxml	Contains BSW module description.

Note: Generated source files need not to be added to your application Makefile. These files will be compiled and linked automatically during the build process.

*Note: Additional steps are required to generate the BSW module description. In EB tresos Studio, select **Project** > **Build Project**, and click **generate_swcd**.*

2.2.4 Dependencies

2.2.4.1 PORT driver

Although the UART driver can be successfully compiled and linked without an AUTOSAR-compliant PORT driver, the latter is required to configure and initialize all ports. Otherwise, the UART driver will show undefined behavior. The PORT driver must be initialized before the UART driver is initialized.

If the UART channel is used for VUART, the corresponding DW trigger routing is also required.

2.2.4.2 MCU driver

The MCU driver must be initialized, and all MCU clock reference points referenced by the hardware units (SCB) via the `UartClockRef` configuration parameter must be activated (via calls of MCU API functions) before initializing the UART driver. See the MCU driver's user guide for details.

Note that the clock (PCLK), prescaler, or PLL settings are controlled by the MCU driver. There are no resources shared with the UART driver. Depending on the configuration, changes in the clock settings may affect the operation of the UART driver.

2.2.4.3 AUTOSAR OS

The AUTOSAR operating system handles the interrupts used by the UART driver. See **2.5.3 Interrupts** for more information.

The counter provided by the operating system is used by the UART driver in the synchronous transmit / reception feature.

If the UART channels are only used for VUART, these setting is not required. Then you should deactivate the corresponding UART configuration parameter.

2.2.4.4 BSW scheduler

The BSW scheduler handles the critical sections that are used by the UART driver.

2.2.4.5 DET

If default error detection is enabled in the UART driver configuration, DET must be installed, configured, and integrated into the application as well.

This driver reports DET error codes as instance 0.

Note: *Even if DET is disabled, parameter check is enabled. Only the notification to DET is disabled, to ensure the capability of safe operation.*

2.2.4.6 DEM

If DEM event reporting is enabled in the UART driver configuration, DEM must be installed, configured, and integrated into the application as well.

To enable DEM support in the UART driver, the `UART_DEM_UNRECOVERABLE_FAILURE` and `UART_DEM_RECOVERABLE_FAILURE` production error must be defined in the DEM configuration in the `UartDemEventParameterRefs` container.

Note: *Even if DEM is disabled, HW error check is enabled. Only the notification to DEM is disabled, to ensure the capability of safe operation.*

2.2.4.7 Error callout handler

The error callout handler is called on every error that is detected, regardless of whether default error detection is enabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via the `UartErrorCalloutFunction` configuration parameter.

2.3 EB tresos Studio configuration interface

The GUI is not part of this delivery. For further information, see the EB tresos Studio for ACG8 user's guide**[5]**.

2.3.1 General configuration

The module comes preconfigured with default settings. You must adapt these to your environment when necessary.

Table 8 General configuration

Parameter	Range	Description
`Config Variant`	`VariantPostBuild`	Specifies the configuration variant (fixed to `VariantPostBuild`).
`UartDemEventParameterRefs`	-	Enables or disables the DEM functionality for the UART driver. If this parameter is disabled, both of the following DEM functionalities are disabled: • `UART_DEM_RECOVERABLE_FAILURE` enables or disables the DEM functionality for recoverable failures, categorized as follows: parity error / frame error, ring buffer overflow • `UART_DEM_UNRECOVERABLE_FAILURE` enables or disables the DEM functionality for unrecoverable failures, categorized as follows: Tx FIFO overflow, Rx FIFO overflow / underflow DW source/destination transfer bus error, DW source/destination address misalignment, DW current descriptor pointer is null, DW active channel is disabled, DW descriptor bus error (when VUART is enabled)
`UartVersionInfoApi`	True (available) False (unavailable)	Specifies whether the `Uart_GetVersionInfo` API function is available.
`UartErrorCalloutFunction`	-	Specifies the name of the error callout function, which is called whenever an error occurs. See **4.7 Required callback functions** for more details and syntax.
`UartDevErrorDetect`	True (enable) False (disable)	Enables or disables the DET functionality for the UART driver.
`UartOsCounterRef`	-	Specifies the reference to the OS counter which is used by the UART driver.
`UartTxNotificationCalloutFunction`	-	Specifies the name of the Tx notification function, which is called whenever a Tx transaction is completed. See **4.7 Required callback functions** for more details and syntax.

Parameter	Range	Description
UartRxNotificationCalloutFunction	-	Specifies the name of the Rx notification function, which is called whenever an Rx transaction is completed. See **4.7 Required callback functions** for more details and syntax.
UartTxErrorNotificationCalloutFunction	-	Specifies the name of the Tx error notification function, which is called whenever a Tx transaction is completed in error. See **4.7 Required callback functions** for more details and syntax.
UartRxErrorNotificationCalloutFunction	-	Specifies the name of the Rx error notification function, which is called whenever an Rx transaction is completed in error. See **4.7 Required callback functions** for more details and syntax.

Table 9 UartIncludeFile

Parameter	Range	Description
UartIncludeFile	-	Specifies the external include files used in the UART driver. Note that the notification function and callout function declarations must be included.

2.3.2 UART configuration

Table 10 UartConfigSet

Parameter	Range	Description
UartConfigSet	-	Specifies the configuration set for the UART driver and its name.
UartChannelConfig	-	Specifies the container name for channel configuration.
UartChannelId	0-255	Specifies the ID for the channel used in the UART driver. It is used as a parameter for API functions. Note that the combination of this parameter and the UartChannelConfig container name must be the same in all configuration sets.
UartScbChannelNumber	SCB0 …	Specifies the SCB resource number. Note that this parameter must be unique within a configuration set. The selected SCB channel must not be used in any other software.

Parameter	Range	Description
UartDefaultCommConfigID	0-254	Specifies the default communication parameter set. The specified setting is used in the Uart_Init function.
UartMaxBuffersize	1-65535[Byte]	Specifies the maximum ring buffer size. It is used by the parameter check in the Uart_SetupEb function. If the corresponding UartChannelConfig is used for VUART, then this parameter is disabled.
UartRxBlockThreshold	0-32767[Byte]	Specifies the threshold value to call the Rx notification. If the received data size reaches this value, UartRxNotificationCalloutFunction is called. It is used only in AUTO receive mode. Note that this value must be less than half of UartMaxBuffersize. If the corresponding UartChannelConfig is used for VUART, then this parameter is disabled.

2.3.3 UART communication configuration

Table 11 UartScbCommConfig

Parameter	Range	Description
UartScbCommConfig	-	Specifies the configuration set for the communication and its name.
UartCommConfigID	0-254	Specifies the ID for the communication setting used in the UART driver. It is used as a parameter for API functions. Note that the combination of this parameter and the UartScbCommConfig container name must be the same in all configuration sets.
UartClockRef	-	Reference to the clock source configuration, which is set in the MCU driver configuration. Note that the runtime system is responsible for activating the selected clock before using the UART driver.
UartClockRefInfo	xxxx [Hz]	Specifies the SCB resource input clock value in Hz, which is referenced as UartClockRef. This value is calculated automatically.
UartBreakLevel	LOW HIGH	Specifies the break detection level. See the HW TRM for more details of this HW feature. If the corresponding UartChannelConfig is used for VUART, then this parameter is disabled.
UartBreakwidth	1-16	Specifies the length in bits of a break detection interval. See the HW TRM for more details of this HW feature.

Parameter	Range	Description
		If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartTxMsbFirst`	LSB MSB	Specifies the first bit for transmission. See the HW TRM for more details of this HW feature.
`UartRxMsbFirst`	LSB MSB	Specifies the first bit for reception. See the HW TRM for more details of this HW feature. If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartTxParity`	NONE ODD EVEN	Specifies the parity configuration for transmission. See the HW TRM for more details of this HW feature.
`UartRxParity`	NONE ODD EVEN	Specifies the parity configuration for reception. See the HW TRM for more details of this HW feature. If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartOpenDrainSelector`	NORMAL OPENDRAIN	Specifies the output mode. See the HW TRM for more details of this HW feature.
`UartFrameErrorDataDrop`	TRUE (drop) FALSE (store)	Specifies the behavior when a frame error is detected. See the HW TRM for more details of this HW feature. If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartParityErrorDataDrop`	TRUE (drop) FALSE (store)	Specifies the behavior when a parity error is detected. See the HW TRM for more details of this HW feature. If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartTxTriggerLevel`	1-127 1-63 (when `UartTxDataWidth` is 9)	Specifies the trigger level for transmission. See the HW TRM for more details of this HW feature.
`UartTxStopbitWidth`	2-8	Specifies the stop period width for transmission. See the HW TRM for more details of this HW feature.
`UartRxStopbitWidth`	2-8	Specifies the stop period width for reception. See the HW TRM for more details of this HW feature. If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartTxDataWidth`	4-9	Specifies the data bit count for transmission. See the HW TRM for more details of this HW feature. (it must be 8 when VUART is enabled)
`UartRxDataWidth`	4-9	Specifies the data bit count for reception. See the HW TRM for more details of this HW feature.

Parameter	Range	Description
		If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartMedianFilterEnable`	TRUE (Enable) FALSE (Disable)	Specifies the usage of the median filter. See the HW TRM for more details of this HW feature. If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartOversamplingfactor`	8-16 4-16 (when VUART is enabled)	Specifies the oversampling factor. See the HW TRM for more details of this HW feature. (it must be 4-16 when VUART is enabled)
`UartCalculatedBaudRate`	512-2000000 [Bps] 512-25000000 [Bps] (when VUART is enabled)	Specifies the calculated baud rate. This value is calculated automatically. (it must be 512-25000000 when VUART is enabled)
`UartTxTimeout`	0-65535 [Ticks]	Specifies the maximum execution period for synchronized transmission. The unit is ticks; it depends on `UartOsCounterRef`. If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.
`UartRxTimeout`	0-65535 [Ticks]	Specifies the maximum execution period for synchronized reception. The unit is ticks; it depends on `UartOsCounterRef`. If the corresponding `UartChannelConfig` is used for VUART, then this parameter is disabled.

2.3.4 UART DW configuration

The UartDWConfig is available, if the corresponding `UartChannelConfig` is used for VUART.

Table 12 UartDWConfig

Parameter	Range	Description
`UartDWConfig`	-	Specifies the container name for the DW channel configuration.
`UartDWChannel`	CPUSS_DW1_TR_IN_<n>	Specifies the DW channel (hardware) resource for UART communication.
`UartDWCRCMode`	NOCRC CRC16 CRC32	Specifies the CRC mode as No CRC (NOCRC), CRC 16 bit (CRC16) and CRC 32 bit (CRC32).
`UartDWCRCPolynomial`	0x0-0xFFFFFFFF (when CRC32 is specified) 0x0-0xFFFF (when CRC16 is specified)	Specifies the CRC Polynomial value. See the HW TRM for more details of this HW feature.

Parameter	Range	Description
`UartDWCRCSeed`	0x0-0xFFFFFFFF (when CRC32 is specified) 0x0-0xFFFF (when CRC16 is specified)	Specifies the CRC Seed value. See the HW TRM for more details of this HW feature.
`UartDWCRCIncludeHeader`	TRUE (include) FALSE (exclude)	Specifies whether the row header is included in the CRC calculation.

2.3.5 Other modules

2.3.5.1 PORT driver

The pins given in section **2.5.1 Ports and pins** must be configured in the PORT driver.

2.3.5.2 MCU driver

The SCB clock must be configured.

2.3.5.3 DET

DET must be configured if the DET functionality is activated.

2.3.5.4 DEM

DEM must be configured if the DEM functionality is activated.

2.3.5.5 AUTOSAR OS

The UART driver's interrupts (listed in section **2.5.3 Interrupts**) must be configured in the AUTOSAR operating system. The counter used by the UART driver must be configured.

Note: *If the corresponding UART channel is used for the VUART driver, the interrupt service for its channel should not be configured. The VUART driver is checking the transmission completion by polling the interrupt flag.*

2.3.5.6 BSW scheduler

The UART driver uses the following services of the BSW scheduler (SchM) to enter and leave critical sections:

```
SchM_Enter_Uart_UART_EXCLUSIVE_AREA_0(void)
```

```
SchM_Exit_Uart_UART_EXCLUSIVE_AREA_0(void)
```

You must ensure that the BSW scheduler is properly configured and initialized before using UART services. The critical sections must prevent any task or interrupt from calling any UART API function or UART interrupt service routine.

2.3.5.7 Software for DMA

The UART driver uses the DMA channel, if the channel is used for VUART.
The UART driver does not modify the global register of the DMA hardware except the CRC register. You must ensure that DMA is globally enabled before using the DMA feature of the UART driver.

2.4 Functional description

The UART driver supports transmit and/or receive communication. These communication modes are operated synchronously or asynchronously.

This chapter describes the basic operation of the UART driver.

If a channel is used for VUART, the channel only supports the Asynchronous transmit via DW. All of this handling is done via VUART API. User no need to call the UART API directly, if it is not required in Chapter **3**.

Other channels support the all feature described as follows.

2.4.1 Overview of UART driver functionality

The UART driver provides the data transmission/reception service.

The communication data is provided from / to the user via a user-specified external buffer.

The UART driver supports the following transmit operation modes:

- Synchronous transmit
- Asynchronous transmit

The UART driver supports the following receive operation modes:

- Automatic receive
- Synchronous receive
- Asynchronous receive

Transmission and reception services are executed independently in the same channel.

2.4.1.1 Data reception overview

After the setting up of the ring buffer, the UART driver starts to receive the data into the ring buffer, whether the synchronous / asynchronous receive API is called or not (AUTO reception).

When the stored data size reaches the configured size (`UartRxBlockThreshold`), the UART driver issues a reception notification via the `UartRxNotificationCalloutFunction`.

If you continue AUTO reception, a ring buffer overflow error will occur. To avoid overflow, set up the ring buffer again, or call the following synchronous / asynchronous receive API to operate the ring buffer.

The asynchronous receive API counts the data in the ring buffer.

When the specified data size is stored in the ring buffer, the UART driver immediately issues a reception notification via the `UartRxNotificationCalloutFunction`.

If not, the UART driver continues to wait for data from the bus, and issues a reception notification when the ring buffer is filled with the specified amount of data.

After receiving the notification, call the `Uart_GetRingBufInfo` to read the data from the ring buffer.

This API reports the address of the data which has not yet been read from the ring buffer. You can access the ring buffer with this address and read the remaining data.

If the data is not needed, you can ignore this procedure.

If you call the asynchronous receive API again, the UART driver discards the data of the previously specified size from the ring buffer.

The synchronous receive API ignores the data already stored in the ring buffer.

This means that the data already stored in the ring buffer will be discarded by this API call.

After data of the specified size is received from the bus, this API returns the result.

Therefore, if you want the newest data from an API call, you can use the synchronous receive API.

On the other hand, if you want to receive continuous data (for example a data stream), you can (continuously) use the asynchronous receive API.

To stop receiving, call the `Uart_CancelReceive` API.

To start receiving again, call the `Uart_StartUartRx` API.

2.4.1.2 State transition

The UART driver has the two state machines. One is for transmission; the other is for reception.

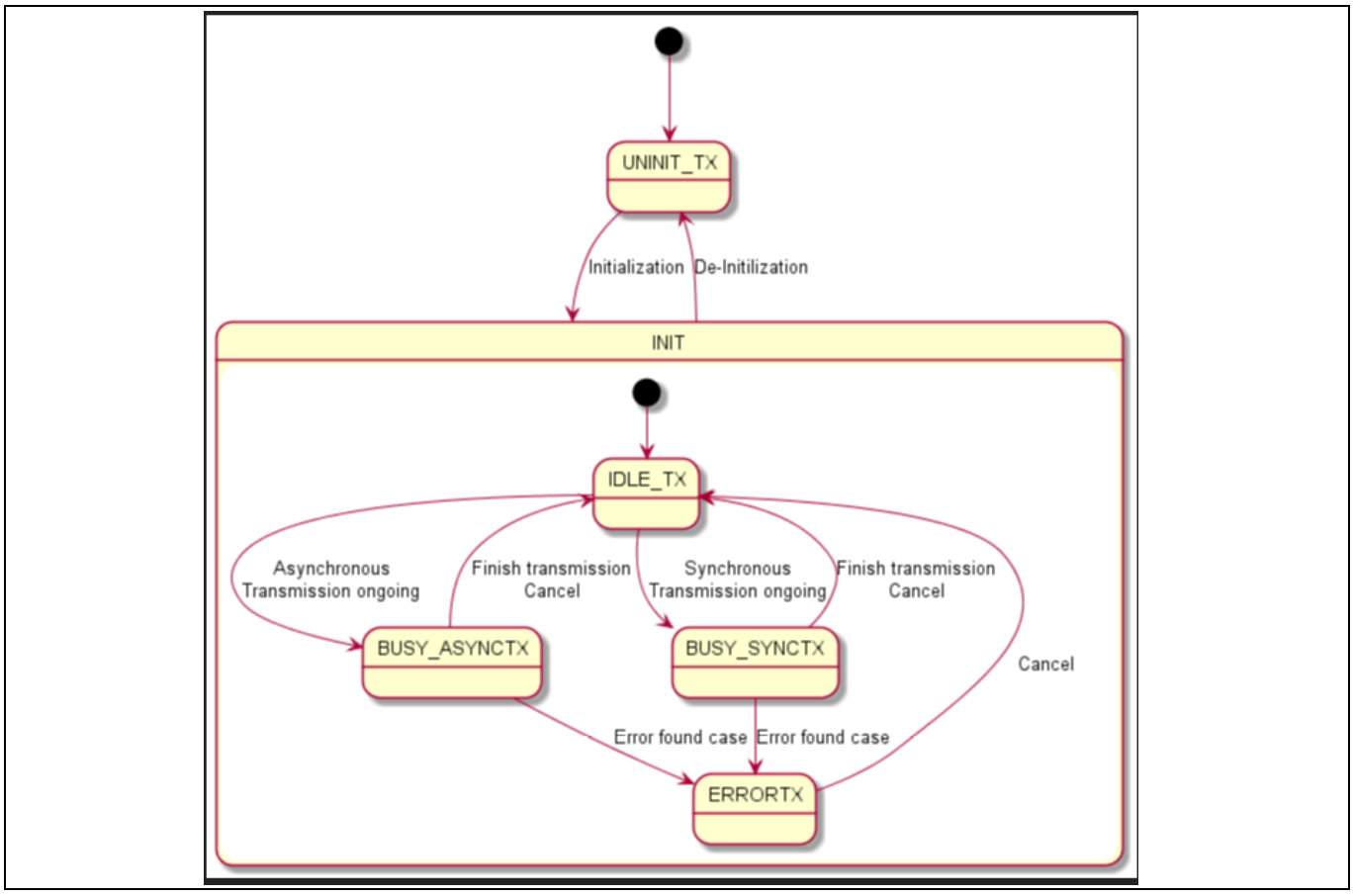

Figure 3 Transmission state machine

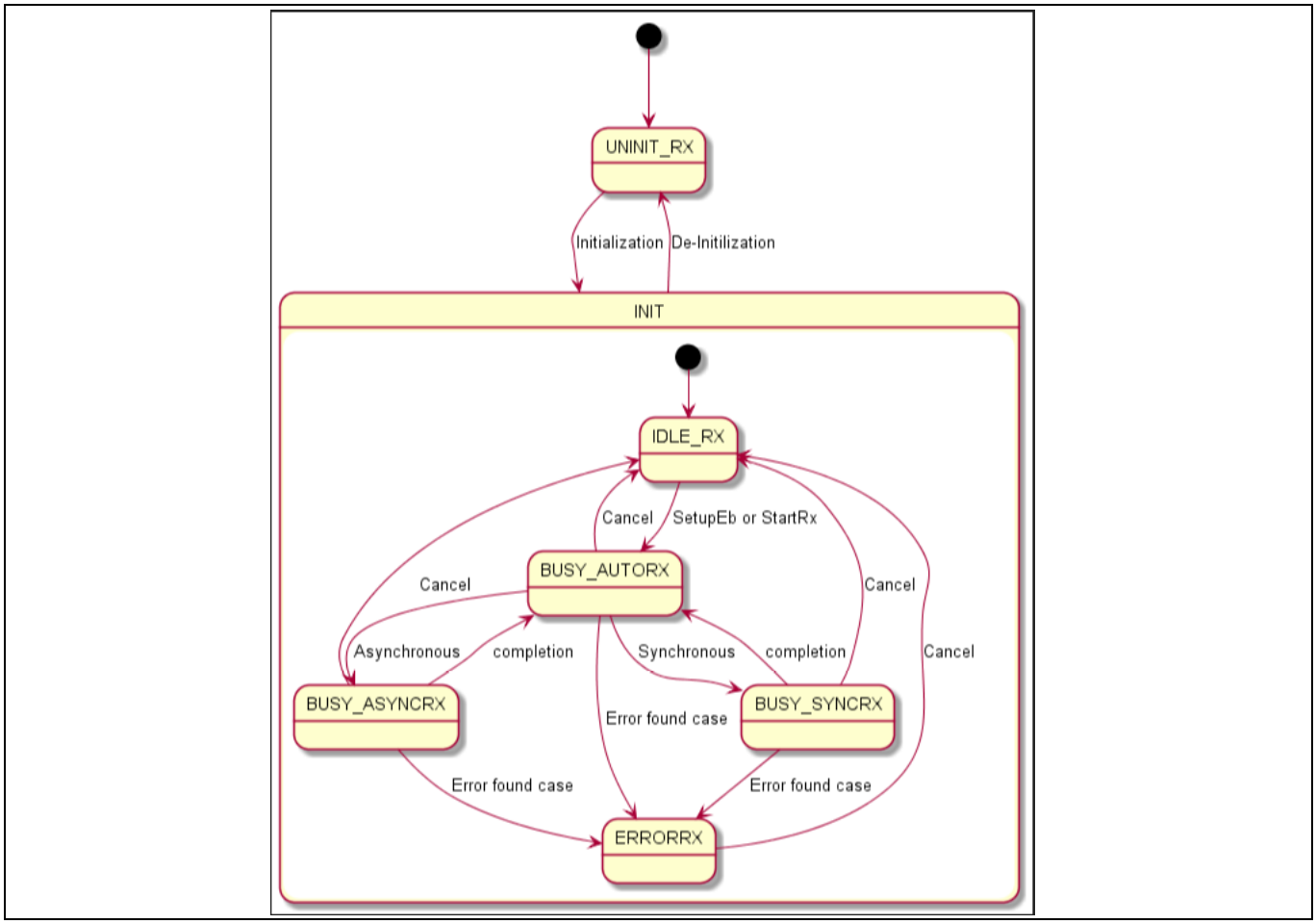

Figure 4 Reception state machine

2.4.2 UART driver functionality and operation

The following sections describe the basic operation of the UART driver.

2.4.2.1 Initialize the UART driver

Before starting the communication, you must initialize the UART driver as follows.

2.4.2.1.1 Call Uart_Init

In this function, the UART driver initializes the configured SCB resources and internal variables.

Code Listing 5 Call Uart_Init

```
001     Uart_Init(&UartConf_UartConfigSet_UartConfigSet_0);
```

See the description of syntax and details in **4.4 Functions**.

2.4.2.2 Prepare the external buffer for reception and start auto reception

Before starting a communication, you must prepare the external buffer (EB). This buffer is used for reception.

2.4.2.2.1 Define the external buffer area for reception

At first, define the external buffer area for reception in your application source code. If you use eight or less data bits (specify eight or less data bits in `UartRxDataWidth`), you can prepare the receive data as follows:

Code Listing 6 Define the external buffer area (eight or less data bits)

```
001     uint8 MyRingBuffer[MY_DATA_SIZE];
002     /* MY_DATA_SIZE is buffer data size (byte) */
```

If you use nine data bits (specify nine bits in `UartRxDataWidth`), you can prepare the receive data as follows:

Code Listing 7 Define the external buffer area (nine data bits)

```
001     Uint16 MyRingBuffer[MY_DATA_SIZE/2];
002     /* MY_DATA_SIZE is buffer data size (byte) */
003     /* array size should be half of MY_DATA_SIZE */
```

Note: The maximum buffer size allowed by the UART driver depends on the `UartMaxBuffersize` *configuration parameter. See the description in **2.3.2 UART configuration**.*

2.4.2.2.2 Call Uart_SetupEb

In this function, the specified buffer area is set as the ring buffer for reception.

Code Listing 8 Call Uart_SetupEb

```
001     Ret = Uart_SetupEb(chId, (uint8*)&MyRingBuffer[0], MY_DATA_SIZE);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the target channel.

The second and the third parameters specify the ring buffer address and size (byte).

Because the type of second parameter is uint8*, the cast to this type is required in case of nine data bits.

After calling this function, the UART driver starts to receive and store data into the specified ring buffer.

Note: *This function is allowed to be called only in IDLE status (Rx must be IDLE).*

2.4.2.3 Start synchronous transmit

2.4.2.3.1 Prepare the transmit data

At first, prepare the transmit data in your application source code. If you use eight or less data bits (specify eight or less data bits in `UartTxDataWidth`), you can prepare the transmit data as follows:

Code Listing 9 **Prepare the transmit data (eight or less data bits)**

```
001         uint8 MyTxBuffer[MY_TX_SIZE];
002         /* store the transmit data into this buffer */
003         /* MY_TX_SIZE is transmit data size (byte) */
```

If you use nine data bits (specify nine bits in `UartTxDataWidth`), you can prepare the transmit data as follows:

Code Listing 10 **Prepare the transmit data (nine data bits)**

```
001         uint16 MyTxBuffer[MY_TX_SIZE/2];
002         /* store the transmit data into this buffer */
003         /* MY_TX_SIZE is transmit data size (byte) */
004         /* array size should be half of MY_TX_SIZE */
```

2.4.2.3.2 Call Uart_SyncTransmit

In this function, the specified data is transmitted to the bus.

Code Listing 11 **Call Uart_SyncTransmit**

```
001         Ret = Uart_SyncTransmit(chId, (uint8*)&MyTxBuffer[0], MY_TX_SIZE);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the target channel.

The second and the third parameters specify the transmit data address and size (byte).

Because the type of second parameter is uint8*, the cast to this type is required in case of nine data bits.

In this function, the UART driver starts to transmit and waits for the end of the transmission.

When all data was transmitted, this function returns the result.

If the execution period is longer than `UartTxTimeout`, this function stops the transmission and returns `E_NOT_OK`.

Note: *This function is allowed to be called only in IDLE status.*

2.4.2.4 Start asynchronous transmit

2.4.2.4.1 Prepare the transmit data

At first, prepare the transmit data in your application source code. If you use eight or less data bits (specify eight or less data bits in `UartTxDataWidth`), you can prepare the transmit data as follows:

Code Listing 12 Prepare the transmit data (eight or less data bits)

```
001         uint8 MyTxBuffer[MY_TX_SIZE];
002         /* store the transmit data into this buffer */
003         /* MY_TX_SIZE is transmit data size (byte) */
```

If you use nine data bits (specify nine bits in `UartTxDataWidth`), you can prepare the transmit data as follows:

Code Listing 13 Prepare the transmit data (nine data bits)

```
001         uint16 MyTxBuffer[MY_TX_SIZE/2];
002         /* store the transmit data into this buffer */
003         /* MY_TX_SIZE is transmit data size (byte) */
004         /* array size should be half of MY_TX_SIZE */
```

2.4.2.4.2 Call Uart_AsyncTransmit

In this function, the UART driver starts to transmit the data to the bus.

Code Listing 14 Call Uart_AsyncTransmit

```
001         Ret = Uart_AsyncTransmit(chId, (uint8*)&MyTxBuffer[0], MY_TX_SIZE);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the target channel.

The second and the third parameters specify the transmit data address and size (byte).

Because the type of second parameter is uint8*, the cast to this type is required in case of nine data bits.

In this function, the UART driver starts to transmit.

The remaining operation is performed by the ISR.

When all data was transmitted, the UART driver notifies the completion of the transmission via the `UartTxNotificationCalloutFunction`.

Note: This function is allowed to be called only in IDLE status.

2.4.2.5 Start synchronous receive

2.4.2.5.1 Prepare the receive data buffer

At first, prepare the receive data buffer in your application source code. (This is not equal to the ring buffer.) If you use eight or less data bits (specify eight or less data bits in `UartRxDataWidth`), you can prepare the receive data as follows:

Code Listing 15 Prepare the receive data buffer (eight or less data bits)

```
001         uint8 MyRxBuffer[MY_RX_SIZE];
002         /* MY_RX_SIZE is receive data size (byte) */
```

If you use nine data bits (specify nine bits in `UartRxDataWidth`), you can prepare the receive data as follows:

Code Listing 16 Prepare the receive data buffer (nine data bits)

```
001         uint16 MyRxBuffer[MY_RX_SIZE/2];
002         /* MY_RX_SIZE is receive data size (byte) */
003         /* array size should be half of MY_RX_SIZE */
```

2.4.2.5.2 Call Uart_SyncReceive

In this function, the driver waits for the reception of data of the specified size from the bus.

Code Listing 17 Call Uart_SyncReceive

```
001         Ret = Uart_SyncReceive(chId, (uint8*)&MyRxBuffer[0], MY_RX_SIZE);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the target channel.

The second and the third parameters specify the receive buffer address and size (byte).

Because the type of second parameter is uint8*, the cast to this type is required in case of nine data bits.

In this function, the UART driver starts to receive new data from the bus, and to copy it to the specified buffer.

This function waits until the specified data size was copied.

When all data was received and copied, this function returns the result, and restarts AUTO reception.

At this time, the data already stored in the ring buffer is ignored.

This means that you cannot read the previously received data after the call of this function.

If the execution period is longer than `UartRxTimeout`, this function stops the reception and returns `E_NOT_OK`.

Note: This function is allowed to be called only in BUSY_AUTO status.

2.4.2.6 Start asynchronous receive

2.4.2.6.1 Call Uart_AsyncReceive

In this function, the UART driver sets the specified data size as a notification criterion.

Code Listing 18 Call Uart_AsyncReceive

```
001         Ret = Uart_AsyncReceive(chId, MY_RX_SIZE);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the target channel.

The second and the third parameters specify the required receive data size.

In this function, the UART driver sets the notification threshold.

When the ring buffer data size reaches the threshold, the UART driver notifies the completion of the receive operation via the `UartRxNotificationCalloutFunction`.

Note: *This function updates the tail pointer (location of the data not read yet). It means that the data previously pointed to by the tail pointer will be recognized as already read. The updated size depends on the previous operation (the size parameter of the Sync/AsyncReceive API, or `UartRxBlockThreshold`).*

Note: *This function is allowed to be called only in BUSY_AUTO status.*

2.4.2.6.2 Call Uart_GetRingBufInfo

Using this function, you can get the ring buffer information.

With this information, you can read the received data from the ring buffer.

Code Listing 19 Call Uart_GetRingBufInfo

```
001         uint8 * TopPtr;
002         uint8 * TailPtr;
003         uint32 RingSize;
004         uint32 EmptySize;
005         Ret = Uart_GetRingBufInfo(chId, &TopPtr, &RingSize, &TailPtr,
   &EmptySize);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the target channel.

The second to fifth parameters provide the output variables.

In this function, the UART driver outputs the ring buffer information to the specified addresses.

- Output the current top address of the ring buffer (which means the newest data location) to the address specified by the second parameter.
- Output the ring buffer size to the address specified by the third parameter.
- Output the current tail address of the ring buffer (which means the location of the data not read yet) to the address specified by the fourth parameter.
- Output the ring buffer empty size (which means the amount of data that can be stored in the ring buffer from now) to the address specified by the fifth parameter.

You can read the received data from the `TailPtr` (fourth parameter) address.

Note: Consider any wraparound if you read the data from the ring buffer.

2.4.2.7 Stop / termination and restart

To stop / terminate the current transmission or reception, use the following functions:

2.4.2.7.1 Call Uart_CancelTransmit

In this function, the UART driver terminates the current transmit operation.

Code Listing 20 Call Uart_CancelTransmit

```
001         Ret = Uart_CancelTransmit(chId);
```

See the description of syntax and details in **4.4 Functions**.

The `chId` parameter specifies the target channel.

In this function, the UART driver stops transmitting and resets the Tx status to IDLE.

If the current status is already IDLE, this function returns with no action.

To restart the transmit operation, use the corresponding transmit API.

Note: This function stops the transmit operation, whether synchronous or asynchronous. If a channel is used for VUART, this function also stops the DW operation.

2.4.2.7.2 Call Uart_CancelReceive

In this function, the UART driver terminates the current receive operation.

Code Listing 21 Call Uart_CancelReceive

```
001         Ret = Uart_CancelReceive(chId);
```

See the description of syntax and details in **4.4 Functions**.

The `chId` parameter specifies the target channel.

In this function, the UART driver stops receiving, and resets the Rx status to IDLE.

If the current status is already IDLE, this function returns with no action.

Note: This function stops the receive operation, whether synchronous, asynchronous, or automatic. To read the stored data from the ring buffer, you should read it before restarting the reception again.

2.4.2.7.3 Call Uart_StartUartRx

In this function, the UART driver restarts the AUTO receive operation.

To restart the Sync/AsyncReceive, call this function before the intended API call. See the state transition in **2.4.1.2 State transition**.

Code Listing 22 Call Uart_StartUartRx

```
001         Ret = Uart_StartUartRx(chId);
```

See the description of syntax and details in **4.4 Functions**.

The `chId` parameter specifies the target channel.

In this function, the UART driver restarts the AUTO receive operation.

Note: This function updates the tail pointer (location of the data not read yet). It means that the data previously pointed to by the tail pointer will be recognized as already read. All previously stored data in the ring buffer are ignored.

2.4.2.8 Get driver information

To know the current driver situation / status, use the following:

2.4.2.8.1 Call Uart_GetCommMode

In this function, the UART driver returns the current communication setting ID.

Code Listing 23 Call Uart_GetCommMode

```
001         CommId = Uart_GetCommMode(chId);
```

See the description of syntax and details in **4.4 Functions**.

The `chId` parameter specifies the target channel.

In this function, the UART driver returns the communication setting ID (specified in the configuration).

2.4.2.8.2 Call Uart_GetTxStatus

In this function, the UART driver return the current Tx status.

Code Listing 24 Call Uart_GetTxStatus

```
001         Status = Uart_GetTxStatus(chId);
```

See the syntax and details in **4.4 Functions**.

The `chId` parameter specifies the target channel.

In this function, the UART driver returns the Tx status (see **2.4.1.2 State transition**).

2.4.2.8.3 Call Uart_GetRxStatus

In this function, the UART driver returns the current Rx status.

Code Listing 25 Call Uart_GetRxStatus

```
001         Status = Uart_GetRxStatus(chId);
```

See the description of syntax and details in **4.4 Functions**.

The `chId` parameter specifies the target channel.

In this function, the UART driver returns the Rx status (see **2.4.1.2 State transition**).

2.4.2.8.4 Call Uart_GetVersionInfo

In this function, the UART driver outputs the version information of the driver.

Code Listing 26 Call Uart_GetVersionInfo

```
001         Std_VersionInfoType VersionInfo;
```

Code Listing 26 Call Uart_GetVersionInfo

```
002         Uart_GetVersionInfo(&VersionInfo);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the address of the output variable.

In this function, the UART driver outputs the driver version information to the address specified by the `VersionInfo` parameter.

2.4.2.8.5 Call Uart_GetTxBufferInfo

In this function, the UART driver outputs the Tx buffer information.

Code Listing 27 Call Uart_GetTxBufferInfo

```
001         uint8 * TopPtr;
002         uint32 Size;
003         Uart_GetTxBufferInfo(chId, &TopPtr, &Size);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the target channel.

The second and the third parameters specify the addresses of the output variables.

In this function, the UART driver outputs the Tx buffer information to the addresses specified by the second and the third parameters.

- Output the current top address of the Tx buffer (which means the next transmit data location) to the address specified by the second parameter.
- Output the Tx buffer remaining size (which means the amount of data not yet transmitted) to the address specified by the third parameter.

You can confirm the progress of the transmit operation by verifying the output.

2.4.2.9 Change the communication setting

If the input clock is changed while the system is running, you can adjust the communication settings (including baud rate setting) with this function.

2.4.2.9.1 Call Uart_SetCommMode

In this function, the UART driver modifies the communication settings.

Code Listing 28 Call Uart_SetCommMode

```
001         Ret = Uart_SetCommMode (chId, CommId);
```

See the description of syntax and details in **4.4 Functions**.

The first parameter specifies the target channel.

The second parameter specifies the target communication setting ID.

In this function, the UART driver changes the communication settings to the configuration set of the specified ID.

Note: *This function changes only the SCB register settings. Note that the input clock for SCB is out of scope. Usually, this input clock is controlled by the MCU module.*

Note: *This function is allowed to be called only in IDLE status. Both transmit and receive operations must be canceled before calling this API.*

2.4.2.10 Deinitialize the driver

2.4.2.10.1 Call Uart_DeInit

In this function, the UART driver resets the related registers and variables to their reset values.

Code Listing 29 Call Uart_DeInit

```
001         Uart_DeInit();
```

See the description of syntax and details in **4.4 Functions**.

Note: *This function is allowed to be called only in IDLE status. Both transmit and receive operations must be canceled before calling this API.*

2.4.3 What should be included

The *Uart.h* file includes all necessary external identifiers. Therefore, your application only needs to include *Uart.h* to make all API functions and data types available.

2.4.4 System initialization

The UART driver must be initialized before use by calling the `Uart_Init` API function.

Before using the UART driver, the following is needed:

- The PORT and MCU module must be initialized
- Prepare other BSW modules (see **2.3.5 Other modules**).

2.4.5 Runtime reconfiguration

To change the configuration set, disable the UART driver (using `Uart_DeInit`). After this, initialize the UART driver (using `Uart_Init`) with another configuration set.

To change a part of the configuration, see section **2.4.2.9 Change the communication setting**. This feature does not require disabling the UART driver.

2.4.6 API parameter checking

The UART driver's services perform regular error checks.

When an error occurs, the error hook routine (configured via `UartErrorCalloutFunction`) is called with the error code, service ID, module ID, and instance ID as parameters.

If default error detection is enabled, all errors are also reported to DET, a central error hook function within the AUTOSAR environment. The checking itself cannot be deactivated for safety reasons.

See section 4 for a description of API functions and associated error codes.

2.4.6.1 Vendor-specific development errors

The UART driver is not included in the AUTOSAR specification, therefore, all parameter error checks are vendor-specific.

Table 13 Vendor-specific development error list

Error	Description
UART_E_UNINIT	An API was called before the initialization of the UART driver.
UART_E_ALREADY_INITIALIZED	Uart_Init was called when the driver was already initialized, without calling Uart_DeInit.
UART_E_TRANSACTION	An API was called when the driver was not in the correct state.
UART_E_OS_TIME_REFUSED	GetCounterValue or GetElapsedValue (OS reference functions) reported an error.
UART_E_PARAM_CONFIG	Uart_Init was called with an invalid parameter (the configuration structure is not found in the configuration set).
UART_E_PARAM_CHANNEL	An API was called with incorrect channel ID (channel ID not found in the configuration set).
UART_E_PARAM_POINTER	An API was called with an invalid pointer (especially: NULL_PTR).
UART_E_PARAM_LENGTH	An API was called with an invalid length/size.
UART_E_PARAM_COMMID	An API was called with an invalid comm ID.

2.4.7 Production errors

There are two types of production errors: recoverable failure and unrecoverable failure.

These errors are reported to the DEM module with the category name, and to the error hook (configured via UartErrorCalloutFunction) with the detailed error code.

2.4.7.1 Recoverable failure

These are temporary errors, which are cleared when the operation is retried.

2.4.7.2 Unrecoverable failure

These errors are typically caused by a hardware failure; if retried, the error may occur again.

Table 14 Production error list

Error	Description
UART_E_RINGBUFER_OVERFLOW	The UART driver detected a ring buffer overflow.
UART_E_HW_RX_PARITY_ERROR	The UART driver (SCB HW) detected an Rx parity error.
UART_E_HW_RX_FRAME_ERROR	The UART driver (SCB HW) detected an Rx frame error.
UART_E_HW_TX_OVERFLOW_ERROR	The UART driver (SCB HW) detected a Tx FIFO overflow.

Error	Description
UART_E_HW_RX_OVERFLOW_ERROR	The UART driver (SCB HW) detected an Rx FIFO overflow.
UART_E_HW_RX_UNDERFLOW_ERROR	The UART driver (SCB HW) detected an Rx FIFO underflow.
UART_E_HW_DW_SRC_BUS_ERROR	The UART driver (DW HW) detected a DW source bus error. (when VUART is enabled for channel)
UART_E_HW_DW_DST_BUS_ERROR	The UART driver (DW HW) detected a DW destination bus error. (when VUART is enabled for channel)
UART_E_HW_DW_SRC_MISAL	The UART driver (DW HW) detected a DW source address misalignment. (when VUART is enabled for channel)
UART_E_HW_DW_DST_MISAL	The UART driver (DW HW) detected a DW destination address misalignment. (when VUART is enabled for channel)
UART_E_DW_HW_CURR_PTR_NULL	The UART driver (DW HW) detected a DW current descriptor pointer is null. (when VUART is enabled for channel)
UART_E_DW_HW_ACTIVE_CH_DIABLED	The UART driver (DW HW) detected a DW active channel is disabled. (when VUART is enabled for channel)
UART_E_DW_HW_DESCR_BUS_ERROR	The UART driver (DW HW) detected a DW descriptor bus error. (when VUART is enabled for channel)

2.4.8 Reentrancy

All services except `UART_Init` and `UART_DeInit` are reentrant if they are executed with different channel IDs. (`Uart_GetVersionInfo` is always reentrant.)

2.4.9 Sleep mode

The UART driver does not provide a dedicated sleep mode.

Note: *All the UART sequences must be completed or stopped before entering the deep sleep mode. The UART operation in deep sleep mode is not guaranteed.*

2.4.10 Debugging support

The UART driver does not support debugging.

2.4.11 Execution time dependencies

The execution time of the API functions depends on certain factors.

Table 15 Execution time dependencies

Affected function	Dependency
Uart_Init() Uart_DeInit()	Number of configured hardware units
Uart_Interrupt_SCBx_cat1 Uart_Interrupt_SCBx_cat2	Number of configured hardware units, trigger level setting
Uart_SyncTransmit Uart_SyncReceive	Send/receive data length

Note: *Execution time depends on input clocks and baud rate too, which are however not described here.*

2.4.12 Deviation from AUTOSAR

UART is not defined in AUTOSAR. Therefore, there is no specific requirement that the UART driver deviates from.

2.5 Related hardware resources

2.5.1 Ports and pins

The UART driver uses the SCB instances of the TRAVEO™ T2G family microcontrollers. The pins listed in **Table 16** are used. Ensure that the pins are correctly set in the PORT driver's configuration.

Table 16 Pins for UART operation

Pin name	Direction	Description
Tx	Output	Transmitter output
Rx	Input	Receiver input

2.5.2 Timer

The UART driver does not use any hardware timers directly. (An OS timer is referenced.)

2.5.3 Interrupts

The interrupt services listed in **Table 17** must be configured correctly for peripherals used by the UART driver. If a peripheral is not used, the corresponding interrupt service should not be used.

Table 17 IRQ vectors and ISR names

IRQ vector	ISR name Cat1	ISR name Cat2
SCB<n> interrupt request	Uart_Interrupt_SCB<*n*>_Cat1	Uart_Interrupt_SCB<*n*>_Cat2

Note: *The OS must associate the named ISRs with the corresponding SCB interrupt.*

Note: *If the corresponding UART channel is used for VUART driver, the interrupt service for its channel should not be configured.*

Note: *For example, if the hardware unit SCB ch.2 is configured,* `Uart_Interrupt_SCB2_Cat2()` *must be called from the (OS-) interrupt service routine of the SCB ch.2 interrupt. For category1 usage, the address of* `Uart_Interrupt_SCB2_ Cat1()` *must be the entry for the SCB ch.2 interrupt in the (OS) interrupt vector table.*

Note: *If the UART interrupt priority is too low, FIFO access may be inhibited by other interrupts. This may cause FIFO overflow or underflow. Thus, you should ensure an appropriate priority of the UART interrupts.*

Note: *On the Arm® Cortex®-M4 CPU, priority inversion of interrupts may occur under specific timing conditions in the integrated system with UART. For more details, see the following errata notice.*

Arm® Cortex®-M4 Software Developers Errata Notice - 838869:
"Store immediate overlapping exception return operation might vector to incorrect interrupt"

If the user application cannot tolerate the priority inversion, a DSB instruction should be added at the end of the interrupt function to avoid the priority inversion.

UART interrupts are handled by an ISR wrapper (handler) in the integrated system. Thus, if necessary, the DSB instruction should be added just before the end of the handler by the integrator.

2.5.4 DMA

The UART driver uses DMA channel, if the channel is used for VUART. The DMA channels can be configured in the UART driver by the user and are enabled or disabled by the UART driver as required. The DMA hardware itself must be enabled globally by the user before the VUART driver can be used for DMA transfer.

When using DMA, ensure that one to one trigger multiplexer is correctly set in the PORT driver's configuration.

Note: *You must ensure that no other software shall not use the configured DMA channel and CRC feature of the DMA hardware during the UART(VUART) driver is working.*

3 VUART driver

This chapter describes VUART specific information.

3.1 Using the VUART driver

This chapter describes all necessary steps to incorporate the VUART driver into your application.

3.1.1 Installation and prerequisites

Same installation and prerequisites are applied as described in **2.1.1 Installation and prerequisites**.

3.1.2 Configuring the VUART driver

The VUART driver can be configured with any AUTOSAR-compliant GCE tool. Save the configuration in a separate file, for example, *Vuart.epc*. For more information about the VUART driver configuration, see **3.3 EB tresos Studio configuration interface**.

3.1.2.1 Configuration outline

Table 18 shows the outline of the VUART driver configuration.

You also need to confirm the **2.1.2 Configuring the UART driver** for underlying UART driver setting.

Table 18 Configuration container and parameters

Container / parameter	Description
`VuartIncludeFile`	Specifies the user-defined include file.
`VuartVersionInfoApi`	Specifies whether the version info API is used.
`VuartSwapNotificationCalloutFunction`	Specifies the name of the Swap notification function.
`VuartDevErrorDetect`	Specifies whether development error detection is used.
`VuartErrorCalloutFunction`	Specifies the name of the error callout function.
`VuartConfigSet`	Container for setting the whole VUART channels configuration.
`VuartChannelConfig`	Container for setting individual channel configuration.
`VuartChannelId`	Specifies the ID used to access a VUART channel.
`VuartChannelConfigRef`	Specifies the UART channel configuration which is used by the VUART driver.
`VuartRowLength`	Specifies the number of rows that the VUART driver converts to pixel images.
`VuartColumnLength`	Specifies the payload column size of each row.
`VuartHeaderLength`	Specifies the size of each row of predefined headers.
`VuartBlankRow`	Specifies the number of blank rows.
`VuartHeaderTableData`	Container for setting the header data for each row.

3.1.3 Adapting your application

To use the VUART driver in your application, include the header files of the VUART, MCU, and PORT drivers by adding the following lines of code to your source code file.

Note: *UART header file is included into VUART header file.*

Code Listing 30 Example for include

```
001         #include "Mcu.h"     /* AUTOSAR MCU Driver */
002         #include "Port.h"    /* AUTOSAR PORT Driver */
003         #include "Vuart.h"   /* VUART Driver */
```

This makes all required functions, data types, and symbolic names known to the application.

To use the VUART driver, you must configure appropriate port pins, and SCB clock settings in the PORT driver, and MCU driver. For detailed information, see **3.5 Related hardware resources**.

You must initialize the MCU, PORT, and start the VUART driver afterwards in the following order:

Code Listing 31 Example for initialize sequence

```
001         Mcu_Init(&Mcu_Config[0]);
002         Port_Init(&Port_Config[0]);
003         …
004         Ret = Vuart_Start(chId);
```

The function `Port_Init()` is called with a pointer to a structure of type `Port_ConfigType`, which is exported by the PORT driver itself.

All required input clocks for the configured hardware units (SCB) must be activated before starting the VUART driver. See **2.3.5.2 MCU driver**.

Note: *UART initialization is done in VUART function. If you use VUART, there is no need to initialize the UART directly in your code.*

Your application must call the `Vuart_MainFunction_Ch[n]` function cyclically. This can either be done by configuring the BSW scheduler accordingly, or by calling the `Vuart_MainFunction_Ch[n]` function from any other cyclic task.

Your application must provide the notification functions that you configured and their declarations. The file containing the declarations must be included using the `VuartIncludeFile` parameter. See the following example of a function declaration and definition for the notification function:

Code Listing 32 Example for declaration

```
001         extern void My_SwapCompleted(uint8 ChannelId);
```

Code Listing 33 Example for definition

```
001         void My_SwapCompleted(uint8 ChannelId)
002         {
003           /* Insert your code here */
004         }
```

This notification function is called from a scheduled function.

3.1.4 Start the build process

The same build process is applied as described in **2.1.4 Start the build process**.

3.1.5 Measuring the stack consumption

Measuring the stack consumption is applied the same as described in **2.1.5 Measuring the stack consumption**.

3.1.6 Memory mapping

The *Vuart_MemMap.h* file in the *$(TRESOS_BASE)/plugins/Vuart_MemmapSample* directory is a sample. This file is replaced by the file generated by the MEMMAP module. The input to the MEMMAP module is described in the *Vuart_Bswmd.arxml* file in the *$(PROJECT_ROOT)/output/generate_swcd/swcd* directory of your project folder.

You also need to confirm the memory mapping for UART in **2.1.6 Memory mapping.**

3.1.6.1 Memory allocation keywords

- `VUART_START_SEC_CODE_ASIL_B / VUART_STOP_SEC_CODE_ASIL_B`

  This memory section type is CODE. All executable code is allocated in this section.

- `VUART_START_SEC_CONST_ASIL_B_UNSPECIFIED / VUART_STOP_SEC_CONST_ASIL_B_UNSPECIFIED`

  This memory section type is CONST. All configuration data is allocated in this section.

- `VUART_START_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED / VUART_STOP_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED`

  This memory section type is VAR. All non-initialized variables without alignment restrictions are allocated in this section.

- `VUART_START_SEC_VAR_INIT_ASIL_B_UNSPECIFIED / VUART_STOP_SEC_VAR_INIT_ASIL_B_UNSPECIFIED`

  This memory section type is VAR. All initialized variables without alignment restrictions are allocated in this section.

3.1.6.2 Restriction for memory allocation for data buffer

The VUART user shall be ensured the data buffer allocation by yourself.
In VUART transaction, the data buffer will be accessed by the CPU and DMA.
Therefore, the data allocation has the following restriction.
The CPU has a private cache that is not shared with the DMA bus master. Therefore, you must ensure that the data accessed by DMA are in uncached memory regions. The VUART driver does not support the memory allocation of DMA-related memory and data buffer to the CPU's tightly coupled memories (TCMs) and internal video RAM (VRAM).

- The section that contains the data buffers used for data transactions: Because the VUART driver is using DMA for data transactions, the section must be allocated to a user-specific memory region configured by the MPU as write-through or non-cacheable.

Note: *These restrictions are applied only to the Cortex®-M7 CPU because it includes TCMs, VRAM and cache. These restrictions do not apply when using the Cortex®-M4 CPU. The areas mentioned above must be accessible through DMA and require 4-byte alignment.*

Note: *This restriction is only applicable for VUART buffer. Not for UART buffer.*

3.2 Structure and dependencies

The VUART driver consists of static, configuration, and generated files.
Underlying UART driver information is described in **2.2 Structure and dependencies.**

3.2.1 Static files

Table 19 Static files

Folder	Description
$(PLUGIN_PATH)=$(TRESOS_BASE)/plugins/Vuart_TS_*	Path to the VUART driver plugin.
$(PLUGIN_PATH)/src	Comprises configuration-dependent source files or derivative-specific files. Each file will be rebuilt when the configuration is changed. All necessary source files will automatically be compiled and linked during the build process, and all include paths will be set if the VUART driver is enabled.
$(PLUGIN_PATH)/include	Basic public include directory needed by the user to include *Vuart.h*.
$(PLUGIN_PATH)/autosar	Contains the AUTOSAR ECU parameter definition with vendor-, architecture-, and derivative-specific adaptations to create a correctly matching parameter configuration for the VUART driver.

3.2.2 Configuration files

The configuration of the VUART driver is done via EB tresos Studio. The file containing the VUART driver's configuration is named *Vuart.xdm* and is in the *$(PROJECT_ROOT)/config* directory. This file serves as the input to generate configuration-dependent source and header files during the build process.

3.2.3 Generated files

During the build process, the following files are generated based on the current configuration description. They are in the *output/generated* subfolder of your project folder.

Table 20 Generated files

Files	Description
include/Vuart_Cfg.h	Contains all symbolic names for the configured VUART channels.
include/Vuart_ExternalInclude.h	Contains the include directives for user-configured external header files.
src/Vuart_Cfg.c	Contains the configuration of the VUART driver.
src/Vuart_MainFunction.c	Provide scheduled function for pixel data transmission row by row.
swcd/Vuart_Bswmd.arxml	Contains BSW module description.

Note: *Generated source files need not to be added to your application Makefile. These files will be compiled and linked automatically during the build process.*

*Note: Additional steps are required to generate the BSW module description. In EB tresos Studio, select **Project** > **Build Project**, and click **generate_swcd**.*

3.2.4 Dependencies

You should confirm the dependency for underlying UART driver described in **2.2.4 Dependencies**.

3.2.4.1 UART driver

The VUART driver provides the data transmission service by using the UART driver. Therefore, the required UART channel referenced by the VUART driver via the configuration parameter `VuartChannelConfigRef` must be configured.

3.2.4.2 DET

If default error detection is enabled in the VUART driver configuration, DET must be installed, configured, and integrated into the application as well.

This driver reports DET error codes as instance 0.

Note: Even if DET is disabled, parameter check is enabled. Only the notification to DET is disabled, to ensure the capability of safe operation.

3.2.4.3 Error callout handler

The error callout handler is called on every error that is detected, regardless of whether default error detection is enabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via the `VuartErrorCalloutFunction` configuration parameter.

3.3 EB tresos Studio configuration interface

The GUI is not part of this delivery. For further information, see the EB tresos Studio for ACG8 user's guide **[5]**.

3.3.1 General configuration

The module comes preconfigured with default settings. You must adapt these to your environment when necessary.

You need to confirm the underlying UART driver configuration described in section **2.3.1**, **2.3.2**, **2.3.3**, **2.3.4**.

Table 21 General configuration

Parameter	Range	Description
`Config Variant`	`VariantPreCompile`	Specifies the configuration variant (fixed to `VariantPreCompile`).
`VuartVersionInfoApi`	True (available) False (unavailable)	Specifies whether the `Vuart_GetVersionInfo` API function is available.
`VuartSwapNotificationCalloutFunction`	-	Specifies swap completed notification callout function name.

Parameter	Range	Description
		See **5.7 Required callback functions** for more details and syntax.
VuartDevErrorDetect	True (enable) False (disable)	Enables or disables the DET functionality for the VUART driver.
VuartErrorCalloutFunction	-	Specifies the name of the error callout function, which is called whenever an error occurs. See **5.7 Required callback functions** for more details and syntax.

Table 22 VuartIncludeFile

Parameter	Range	Description
VuartIncludeFile	-	Specifies the external include files used in the VUART driver. Note that the notification function and callout function declarations must be included.

3.3.2 VUART configuration

Table 23 VuartConfigSet

Parameter	Range	Description
VuartConfigSet	-	Specifies the configuration set for the VUART driver and its name.
VuartChannelConfig	-	Specifies the container name for channel configuration.
VuartChannelId	0-255	Specifies the ID for the channel used in the VUART driver. It is used as a parameter for API functions. Note that the combination of this parameter and the VuartChannelConfig container name must be the same in all configuration sets.
VuartRowLength	1-1024	Specifies the number of rows that the VUART driver converts to pixel images.
VuartColumnLength	1-65535[Byte]	Specifies the payload column size of each row converted by the VUART driver for pixel images.
VuartHeaderLength	1-4[Byte]	Specifies the size of each row of predefined headers to be converted by the VUART driver.
VuartBlankRow	0-255	Specifies the number of blank rows.

Parameter	Range	Description
`VuartChannelConfigRef`	-	Reference to the `UartChannelConfig` source configuration, which is set in the UART driver configuration.
`VuartHeaderTableData`	0x0-0xFFFFFFFF	Specifies the header data for each row. The range depends on `VuartHeaderLength`.

3.3.3 Other modules

3.3.3.1 UART driver

The UART driver must be configured. See section **2.3.1**, **2.3.2**, **2.3.3**, **2.3.4**.

3.3.3.2 PORT driver

The pins given in section **3.5.1 Ports and pins** must be configured in the PORT driver. The trigger multiplexer given in **2.5.4 DMA** must be configured in the PORT driver.

3.3.3.3 DET

DET must be configured if the DET functionality is activated.

3.4 Functional description

The VUART driver supports asynchronous transmission for pixel image data. This communication is operated via the UART driver.

This chapter describes the basic operation of the VUART driver.

3.4.1 Overview of VUART driver functionality

The VUART driver provides the data transmission service.

The following figure shows the VUART entire system.

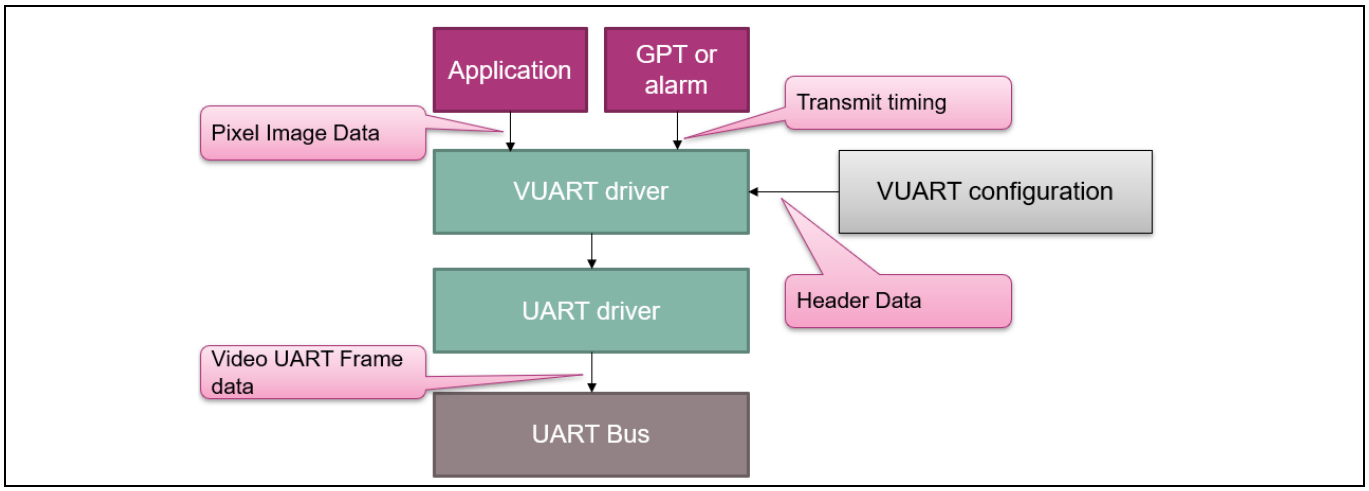

Figure 5 VUART entire system

The application provides the pixel image data to the VUART driver. The VUART driver converts the pixel image data to frame data format, and transmit it to UART bus via the UART driver.

The pixel image data is provided from the application via a user-specified external buffer per one frame.

The VUART driver supports only the Asynchronous transmit operation mode.

3.4.1.1 Double buffering mechanism

The VUART driver has the double buffering mechanism.

During transmission of one frame data, the user can put the next pixel image data to the VUART driver (such as a transmit reservation).

After transmitting the current frame data, the VUART driver will swap the transmit target pointer to next pixel image.

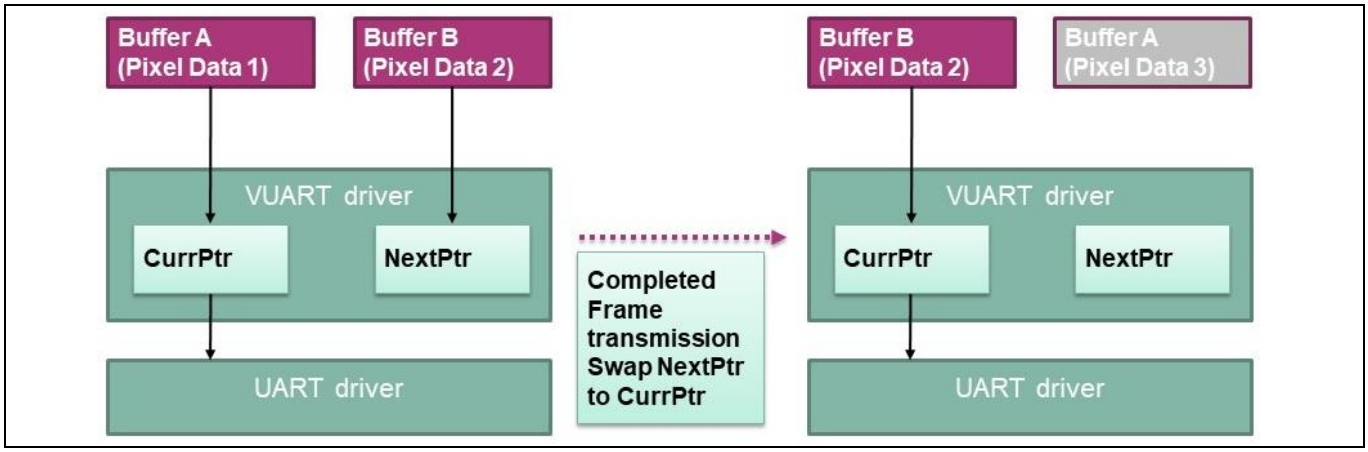

Figure 6 Double buffering mechanism

3.4.1.2 Frame data structure

Figure 7 and **Figure 8** shows the frame data structure which is transmitted by the VUART driver. **Figure 7** depicts the use case for one row. **Figure 8** depicts the use case for multiple row. The frame data is divided into row, a row is transmitted by each transmit trigger (for example, BSW scheduler or GPT feature).

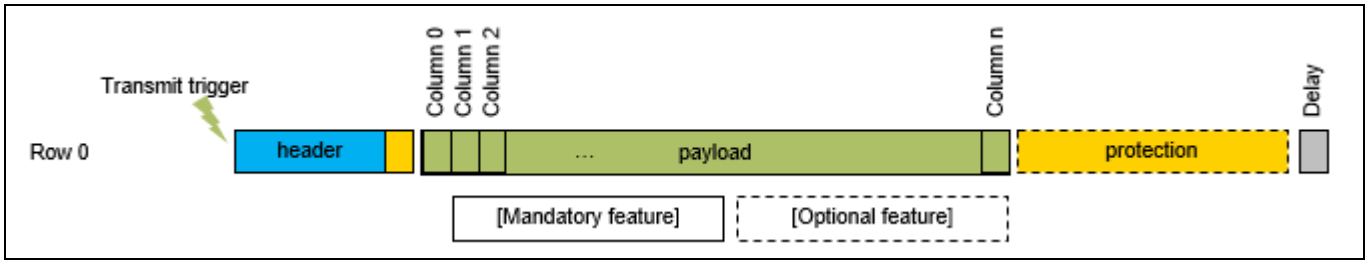

Figure 7 VUART frame data structure (one row)

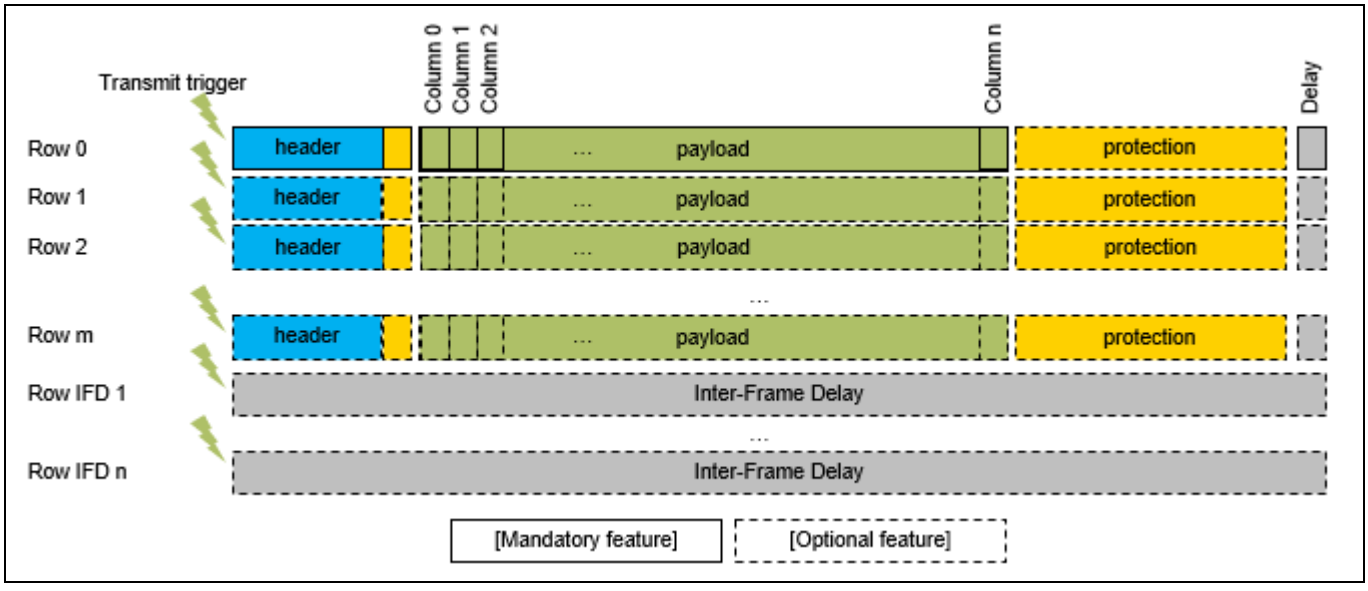

Figure 8 VUART frame data structure (multiple row)

Details of frame data are as follows:

- Header: Header data is transmitted first. All header data must be provided via the configuration parameter `VuartHeaderTableData`. The size of each header data is specified via the configuration parameter `VuartHeaderLength`. For example, this header data can be used for row address and parity bit.

Note: *The VUART driver does not care about the protection (for example, parity bit) of header data. The header data with protection must be configured by the user.*

- Payload: This is a set of pixel image data for one row. Payload consist of pixel data (1 byte per pixel) from column 0 to n. Whole payload data from row 0 to m are specified by calling the `Vuart_UpdateBuffer` function. The size of row is specified via the configuration parameter `VuartRowLength`. The size of column is specified via the configuration parameter `VuartColumnLength`.
- Protection: The CRC can be used for protection. The CRC is calculated by using DMA feature automatically. The CRC must be configured via the configuration `UartDWConfig` of the UART driver. The CRC mode (No CRC or CRC 16 bit or CRC 32 bit) can be specified via the configuration parameter `UartDWCRCMode`. The CRC polynomial value can be specified via the configuration parameter `UartDWCRCPolynomial`. The CRC seed value can be specified via the configuration parameter `UartDWCRCSeed`. In addition, the user can specify whether the row header is included in the CRC calculation via the configuration parameter `UartDWCRCIncludeHeader`.
- Delay of each row: This is an interval between each row transmission.

Note: *The VUART driver does not care about this delay. This delay length depends on the transmit trigger term and payload data length, the baud rate of transmission.*

- Inter-Frame delay: This is an interval between each frame transmission. This delay can be specified via the configuration parameter `VuartBlankRow` as the size of blank row.

Note: *The VUART driver does not transmit any data during inter-frame delay.*

3.4.1.3 State transition

The VUART driver has the one state machine for transmission. Before starting the transmission, the state shall be `VUART_START` by calling `Vuart_Start`. When `Vuart_MainFunction_Ch[n]` is called from VUART_START, the state transitions to VUART_START_MAIN. In this state, calling `Vuart_Stop` is declined. After finishing `Vuart_MainFunction_Ch[n]`, the state transitions to VUART_START.

To stop the transmission, `Vuart_Stop` shall be called. When `Vuart_Stop` is called from VUART_START, the state transitions to VUART_STOPPING. In this state, calling `Vuart_MainFunction_Ch[n]` is declined. After finishing `Vuart_Stop`, the state transitions to VUART_STOP. In case of transmission error, you shall stop the transmission by calling `Vuart_Stop`, and then you can restart again.

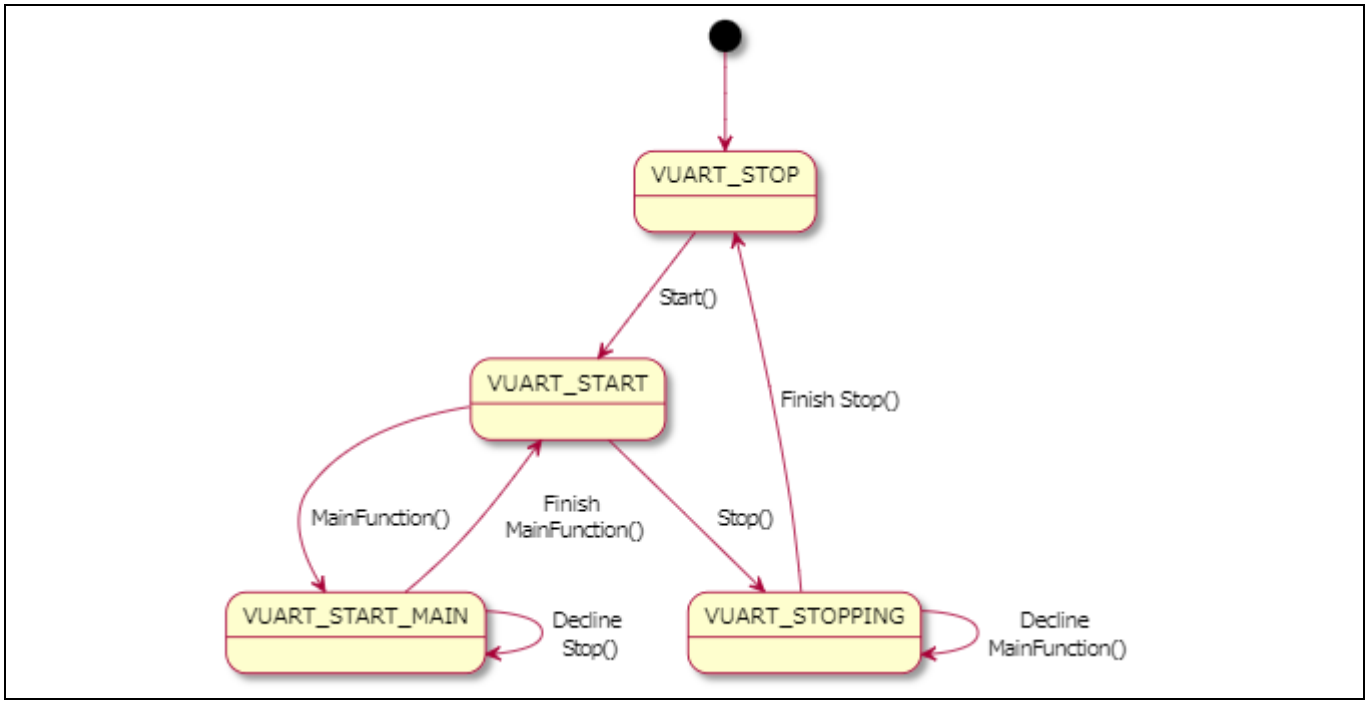

Figure 9 Transmission state machine

3.4.1.4 Transmission overview

This section describes the overview of how to transmit the frame data.

At first, let us see one row use case.

- Provide pixel image data to VUART: `Vuart_UpdateBuffer` can be called to provide pixel image data.
- Convert pixel image data to frame data: `Vuart_MainFunction_Ch0` converts pixel image data to frame data.
- Transmit the frame data: `Vuart_MainFunction_Ch0` transmit the frame data.
- Swap to next buffer: `Vuart_MainFunction_Ch0` swap to next buffer, and notify via `Vuart_SwapCompleted`.

At first, the user provides a pixel image data as primary buffer data to the VUART driver.

Next, the user calls the `Vuart_MainFunction_Ch0` to proceed the frame data transmission.

During transmission, the user can provide the next pixel image data as secondary buffer data.

After swap notification, you can modify the contents of primary buffer data.

After swap notification, if you call the `Vuart_MainFunction_Ch0`, the secondary buffer data will be transmitted.

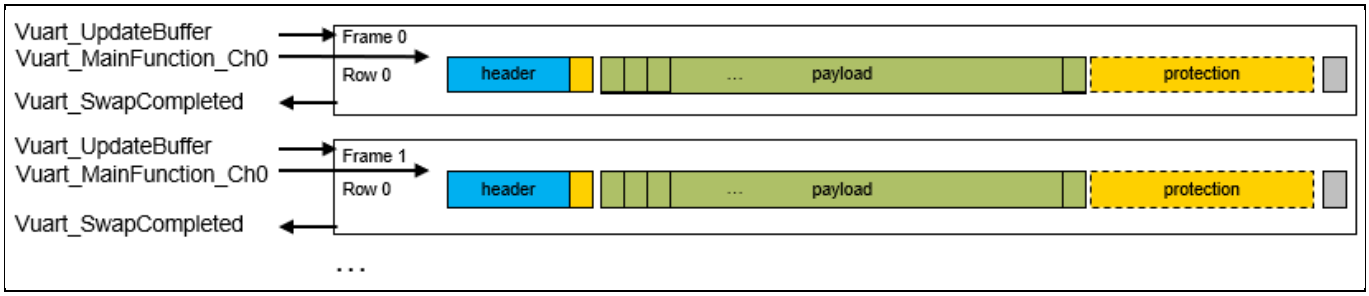

Figure 10 Transmit use case (one row)

Now let us see multiple row use case.

- Provide pixel image data to VUART: `Vuart_UpdateBuffer` can be called to provide pixel image data.
- Convert pixel image data to frame data: `Vuart_MainFunction_Ch0` converts pixel image data to frame data per row.
- Transmit the frame data: `Vuart_MainFunction_Ch0` transmit the frame data per row.
- Inter-Frame Delay: `Vuart_MainFunction_Ch0` acts no operation (just count up the row).
- Swap to next buffer: `Vuart_MainFunction_Ch0` swap to next buffer, and notify via `Vuart_SwapCompleted`.

At first, the user provides a pixel image data as primary buffer data to the VUART driver.

Next, the user calls the `Vuart_MainFunction_Ch0` to proceed the frame data transmission per row.

During transmission, the user can provide the next pixel image data as secondary buffer data.

If the inter-frame delay is configured, the `Vuart_MainFunction_Ch0` just count-up the row number (no transmit operation).

When the all frame data (incl. inter-frame delay) is transmitted, swap notification is called.

After swap notification, you can modify the contents of primary buffer data.

After swap notification, if you call the `Vuart_MainFunction_Ch0`, the secondary buffer data will be transmitted.

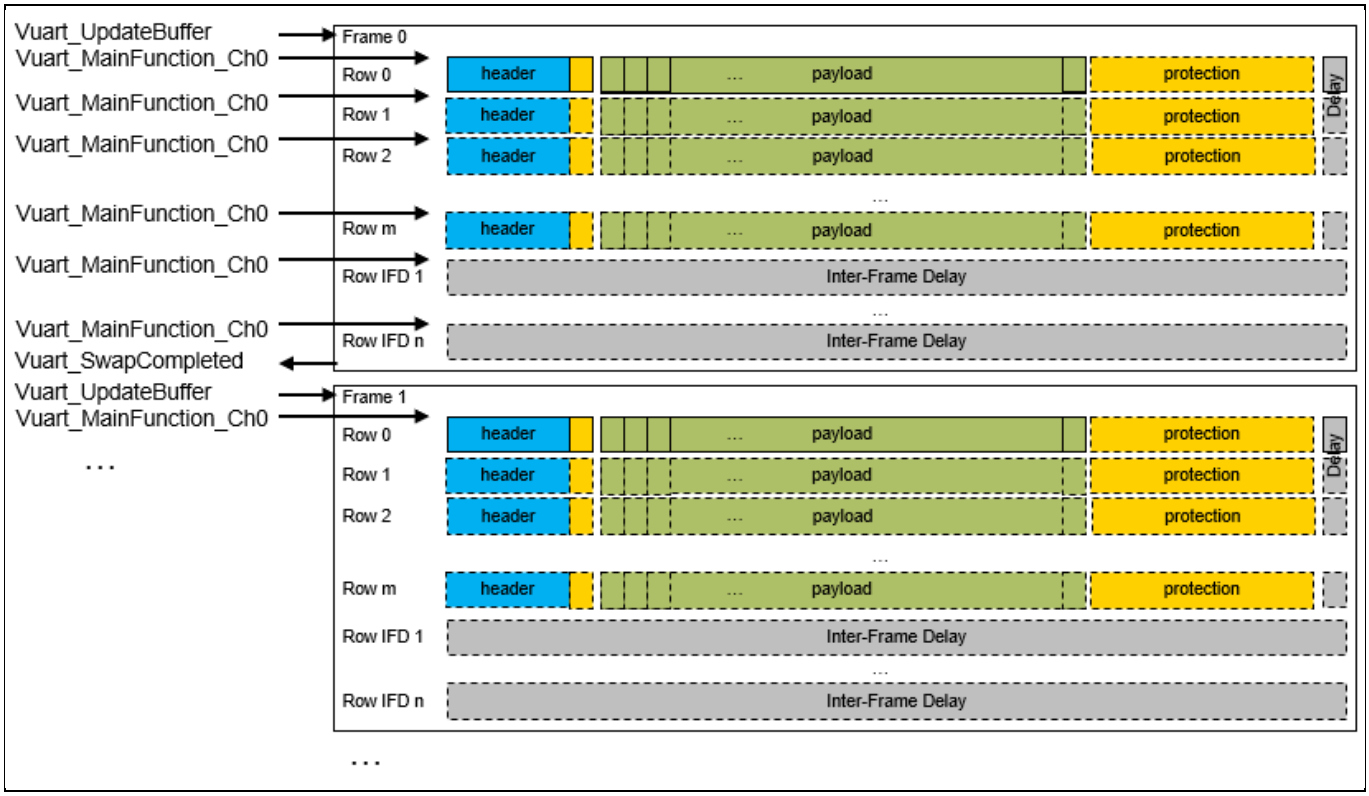

Figure 11 Transmit use case (multiple row)

3.4.2 VUART driver functionality and operation

The following sections describe the basic operation of the VUART driver.

3.4.2.1 Start Vuart

3.4.2.1.1 Call Vuart_Start

In this function, the VUART driver prepares the transmission operation.

Code Listing 34 Call Vuart_Start

```
001         Ret = Vuart_Start(chId);
```

See the description of syntax and details in **5.4 Functions**.

The `chId` parameter specifies the target channel.

In this function, the VUART driver prepares the transmission operation. To enable the transmission operation, this function calls `Uart_Init`, if the UART driver is not initialized.

3.4.2.1.2 Call Vuart_UpdateBuffer

In this function, the VUART driver updates the buffer pointer of the image frame data for subsequent transmission.

Code Listing 35 Call Vuart_UpdateBuffer

```
001         Ret = Vuart_UpdateBuffer(chId, BufferPtr);
```

See the description of syntax and details in **5.4 Functions**.

The first parameter specifies the target channel.

The second parameter specifies the transmit data address for subsequent transmission.

The VUART driver can keep only the buffer pointer for currently ongoing transmission (current buffer) and the buffer pointer for subsequent transmission (next buffer).

In this function, the VUART driver updates the next buffer pointer for subsequent transmission.

3.4.2.1.3 Call Vuart_MainFunction_Ch[n]

In this function, the VUART driver starts to transmit the payload for each row.

Code Listing 36 Call Vuart_MainFunction_Ch[n]

```
001         Ret = Vuart_MainFunction_Ch[n]();
```

See the description of syntax and details in **5.5 Scheduled functions**.

In this function, the VUART driver starts to transmit 1 row payload data.

The remaining operation is performed by the DW.

Each time `Vuart_MainFunction_Ch[n]` is called periodically, 1 row payload data transmission is performed, and then the blank row without transmission is counted.

After all row payload data and blank row are completed, the VUART driver updates the buffer pointer to next data buffer pointer and notifies the completion via the `VuartSwapNotificationCalloutFunction`.

Note: This function shall be called periodically in sufficient duration to ensure the transmission for one row payload data.

3.4.2.2 Get driver information

To know the current driver information, use the following:

3.4.2.2.1 Call Vuart_GetVersionInfo

In this function, the VUART driver outputs the version information of the driver.

Code Listing 37 Call Vuart_GetVersionInfo

```
001         Std_VersionInfoType VersionInfo;
002         Vuart_GetVersionInfo(&VersionInfo);
```

See the description of syntax and details in **5.4 Functions**.

The first parameter specifies the address of the output variable.

In this function, the VUART driver outputs the driver version information to the address specified by the `VersionInfo` parameter.

3.4.2.2.2 Call Vuart_GetCurrentRow

In this function, the VUART driver outputs the current row number.

Code Listing 38 Call Vuart_GetCurrentRow

```
001         uint16 CurrentRow;
002         Vuart_GetCurrentRow(chId, &CurrentRow);
```

See the description of syntax and details in **5.4 Functions**.

The first parameter specifies the target channel.

The 2nd parameter specifies the address of the output variable.

In this function, the VUART driver outputs the current row number (which means the row number currently being transferred or the blank row number currently being proceeded) to the address specified by the 2nd parameter.

If the blank row number is being processed, the output current row number will include the row numbers that have already been transferred (transferred row number + current blank row number).

You can confirm the progress of the transmit operation by verifying the output.

3.4.2.3 Stop / termination and restart

To stop / terminate the current transmission, use the following:

3.4.2.3.1 Call Vuart_Stop

In this function, the VUART driver terminates the current transmit operation.

Code Listing 39 Call Vuart_Stop

```
001         Ret = Vuart_Stop(chId);
```

See the description of syntax and details in **5.4 Functions**.

The `chId` parameter specifies the target channel.

In this function, the VUART driver stops transmitting and resets the Tx status to IDLE by calling the `Uart_CancelTransmit` API function.

If the current status is UNKNOWN status, this function returns with no action.

To restart the transmit operation, use the corresponding transmit API.

3.4.3 What should be included

The *Vuart.h* file includes all necessary external identifiers. Therefore, your application only needs to include *Vuart.h* to make all API functions and data types available.

3.4.4 System initialization

The VUART driver and the UART driver must be initialized before use by calling the `Vuart_Start` API function. If the UART driver is not initialized, the `Uart_Init` is called from `Vuart_Start`.

Before using the VUART driver, the following modules are required:

- The PORT and MCU module must be initialized
- Prepare other BSW modules (see **3.3.3 Other modules**).

3.4.5 Runtime reconfiguration

The VUART driver does not provide this feature.

3.4.6 API parameter checking

The VUART driver's services perform regular error checks.

When an error occurs, the error hook routine (configured via `VuartErrorCalloutFunction`) is called with the error code, service ID, module ID, and instance ID as parameters.

If default error detection is enabled, all errors are also reported to DET, a central error hook function within the AUTOSAR environment. The checking itself cannot be deactivated for safety reasons.

See section **5** for a description of API functions and associated error codes.

3.4.6.1 Vendor-specific development errors

The VUART driver is not included in the AUTOSAR specification. Therefore, all parameter error checks are vendor-specific.

Table 24 Vendor-specific development error list

Error	Description
`VUART_E_ALREADY_STARTED`	VUART is already started.
`VUART_E_PARAM_POINTER`	Pointer out of range.
`VUART_E_PARAM_CHANNEL`	Channel ID out of range.
`VUART_E_TRANSACTION`	Not called in IDLE status.
`VUART_E_NOTSTARTED`	VUART is not started.

3.4.7 Production errors

There is no production error in the VUART driver, but the UART driver's service used by the VUART driver requires the DEM. For detailed information, see **2.4.7 Production errors**.

3.4.8 Reentrancy

All services except `Vuart_Start` and `Vuart_MainFunction_Ch[n]` are reentrant if they are executed with different channel IDs. (`Vuart_GetVersionInfo` is always reentrant.)

3.4.9 Sleep mode

The VUART driver does not provide a dedicated sleep mode.

Note: All the VUART sequences must be completed or stopped before entering the deep sleep mode. The VUART operation in deep sleep mode is not guaranteed.

3.4.10 Debugging support

The VUART driver does not support debugging.

3.4.11 Execution time dependencies

The execution time of the API functions depends on certain factors.

Table 25 Execution time dependencies

Affected function	Dependency
`Vuart_Start()`	Number of configured hardware units of the UART driver, because `Vuart_Start()` calls `Uart_Init()` as required
`Vuart_MainFunction_Ch[n]`	Number of transmission data, data transmission setting

3.4.12 Deviation from AUTOSAR

VUART is not defined in AUTOSAR. Therefore, there is no specific requirement that the VUART driver deviates from.

3.5 Related hardware resources

3.5.1 Ports and pins

The VUART driver uses the SCB instances of the TRAVEO™ T2G family microcontrollers. The pins listed in **Table 26** are used. Ensure that the pins are correctly set in the PORT driver's configuration.

Table 26 Pins for VUART operation

Pin name	Direction	Description
Tx	Output	Transmitter output

3.5.2 Timer

The VUART driver does not use any hardware timers.

3.5.3 Interrupts

The VUART driver does not use any interrupt.

4 Appendix A: UART API and external definitions

4.1 Include files

The *Uart.h* file is the only file that needs to be included to use the functions of the UART driver.

4.2 Data types

4.2.1 Uart_ChannelIdType

Table 27 Uart_ChannelIdType

Type	uint8
Description	Type of channel ID. This type is used to specify a channel.

4.2.2 Uart_BufferType

Table 28 Uart_BufferType

Type	uint8
Description	Type of buffer element. This type is used for buffer reference pointers.

4.2.3 Uart_BufferSizeType

Table 29 Uart_BufferSizeType

Type	uint32
Description	Type of buffer size (in bytes). This type is used to specify a buffer size.

4.2.4 Uart_CommIdType

Table 30 Uart_CommIdType

Type	uint8
Description	Type of communication settings ID. This type is used to specify the communication settings.

4.2.5 Uart_ChannelTxStatusType

Table 31 Uart_ChannelTxStatusType

Type	`typedef enum`
	`{`
	`  UART_UNKNOWN_STATUS_TX  = 0,`
	`  UART_UNINIT_TX  = 1,`
	`  UART_IDLE_TX  = 2,`
	`  UART_BUSY_ASYNCTX  = 3,`
	`  UART_BUSY_SYNCTX  = 4,`
	`  UART_ERRORTX  = 5`
	`} Uart_ChannelTxStatusType;`

Description	Type of transmission status. `UART_UNKNOWN_STATUS_TX`: UART driver cannot judge the transmission status. `UART_UNINIT_TX`: UART driver is uninitialized. `UART_IDLE_TX`: UART driver is not executing a transmission operation. `UART_BUSY_ASYNCTX`: UART driver is operating in asynchronous transmission. `UART_BUSY_SYNCTX`: UART driver is operating in synchronous transmission. `UART_ERRORTX`: UART driver detected a transmission error.

4.2.6 Uart_ChannelRxStatusType

Table 32 Uart_ChannelRxStatusType

Type	`typedef enum`
	`{`
	`  UART_UNKNOWN_STATUS_RX  = 0,`
	`  UART_UNINIT_RX  = 1,`
	`  UART_IDLE_RX  = 2,`
	`  UART_BUSY_AUTORX  = 3,`
	`  UART_BUSY_ASYNCRX  = 4,`
	`  UART_BUSY_SYNCRX  = 5,`
	`  UART_ERRORRX  = 6`
	`} Uart_ChannelRxStatusType;`

Description	Type of reception status. `UART_UNKNOWN_STATUS_RX`: UART driver cannot judge the reception status. `UART_UNINIT_RX`: UART driver is uninitialized. `UART_IDLE_RX`: UART driver is not executing a reception operation. `UART_BUSY_AUTORX`: UART driver is operating in asynchronous reception without the API. `UART_BUSY_ASYNCRX`: UART driver is operating in asynchronous reception. `UART_BUSY_SYNCRX`: UART driver is operating in synchronous reception. `UART_ERRORRX`: UART driver detected a reception error.

4.2.7 Uart_ConfigType

Table 33 Uart_ConfigType

Type	`typedef struct`
	`{`
	`  /* Stores pointer to Uart_ScbChannelConfigType table. */`
	`  P2CONST(Uart_ScbChannelConfigType, TYPEDEF, TYPEDEF) Uart_ScbChannelConfigsPtr;`
	`  /* Pointer Scb Number Index. */`
	`  P2CONST(Uart_ScbNumIndexType, TYPEDEF, TYPEDEF) Uart_ScbNumberIndexPtr;`
	`  /* Stores the number of channel. */`
	`  CONST(uint8, TYPEDEF) NumberOfChannel;`
	`} Uart_ConfigType;`

Description	The type of external data structure containing the initialization data for the UART driver.

4.2.8 Uart_HeaderType

Table 34 Uart_HeaderType

Type	uint8
Description	Type of header element. This type is used for header buffer reference pointers.

4.3 Constants

4.3.1 Error codes

A service may return one of the following error codes if default error detection is enabled.

Table 35 Error codes

Name	Value	Description
UART_E_UNINIT	0x12	No initialization done.
UART_E_ALREADY_INITIALIZED	0x13	Initialization is already done.
UART_E_TRANSACTION	0x14	Not called in IDLE status.
UART_E_RINGBUFER_OVERFLOW	0x15	Ring buffer has been overwritten without a read.
UART_E_OS_TIME_REFUSED	0x16	OS reference function returned an error code.
UART_E_PARAM_CONFIG	0x17	Configuration pointer out of range.
UART_E_PARAM_CHANNEL	0x18	Channel ID out of range.
UART_E_PARAM_POINTER	0x19	Pointer out of range.
UART_E_PARAM_LENGTH	0x1A	Length out of range.
UART_E_PARAM_COMMID	0x1B	Communication setting ID out of range.
UART_E_HW_RX_PARITY_ERROR	0x1C	Parity error detected.
UART_E_HW_RX_FRAME_ERROR	0x1D	Frame error detected.
UART_E_HW_TX_OVERFLOW_ERROR	0x1E	Tx FIFO overflow.

Name	Value	Description
UART_E_HW_RX_OVERFLOW_ERROR	0x1F	Rx FIFO overflow.
UART_E_HW_RX_UNDERFLOW_ERROR	0x21	Rx FIFO underflow.
UART_E_HW_DW_SRC_BUS_ERROR	0x23	DW source bus error.
UART_E_HW_DW_DST_BUS_ERROR	0x24	DW destination bus error.
UART_E_HW_DW_SRC_MISAL	0x25	DW source address misalignment.
UART_E_HW_DW_DST_MISAL	0x26	DW destination address misalignment.
UART_E_DW_HW_CURR_PTR_NULL	0x27	DW current descriptor pointer is null.
UART_E_DW_HW_ACTIVE_CH_DIABLED	0x28	DW active channel is disabled.
UART_E_DW_HW_DESCR_BUS_ERROR	0x29	DW descriptor bus error.

4.3.2 Version / module information

Table 36 Version / module information

Name	Value	Description
UART_SW_MAJOR_VERSION	See release notes	Vendor-specific major version number
UART_SW_MINOR_VERSION	See release notes	Vendor-specific minor version number
UART_SW_PATCH_VERSION	See release notes	Vendor-specific patch version number
UART_MODULE_ID	255	Module ID
UART_VENDOR_ID	66	Vendor ID

4.3.3 API service ID

The following API service IDs are used when reporting errors via DET or via the error callout function.

Table 37 API service ID

Name	Value	API name
UART_API_INIT	0x0F	Uart_Init
UART_API_DEINIT	0x10	Uart_DeInit
UART_API_SET_COMM	0x11	Uart_SetCommMode
UART_API_GET_COMM	0x12	Uart_GetCommMode
UART_API_GET_TXSTATUS	0x37	Uart_GetTxStatus
UART_API_GET_RXSTATUS	0x38	Uart_GetRxStatus
UART_API_SETUP_EB	0x3A	Uart_SetupEb
UART_API_SYNC_TX	0x40	Uart_SyncTransmit
UART_API_SYNC_RX	0x3C	Uart_SyncReceive
UART_API_ASYNC_TX	0x41	Uart_AsyncTransmit
UART_API_ASYNC_RX	0x3D	Uart_AsyncReceive
UART_API_CANCEL_TX	0x43	Uart_CancelTransmit
UART_API_CANCEL_RX	0x3F	Uart_CancelReceive
UART_API_INTERRUPT_SCB	0x5B	Uart_Interrupt_SCB<n>_Cat1,

Name	Value	API name
		Uart_Interrupt_SCB<n>_Cat2
UART_API_START_RX	0x3B	Uart_StartUartRx
UART_API_GET_RINGBUF_INFO	0x3E	Uart_GetRingBufInfo
UART_API_GET_VERSION_INFO	0x39	Uart_GetVersionInfo
UART_API_GET_TXBUF_INFO	0x42	Uart_GetTxBufferInfo
UART_API_GET_TX_INT_STATUS	0x45	Uart_GetTxIntStatus
UART_API_ASYNC_TX_HEADER	0x44	Uart_AsyncTransmitWithHeader

4.4 Functions

4.4.1 Uart_Init

Table 38 Uart_Init

Syntax	`FUNC(void, UART_CODE) Uart_Init`
	`(`
	`  P2CONST(Uart_ConfigType, AUTOMATIC, UART_APPL_CONST) ConfigPtr`
	`)`

Service ID	0x0F
Sync/Async	Sync
Reentrancy	Non-reentrant
Parameters (in)	`ConfigPtr`: pointer to configuration structure
Parameters (out)	None
Return value	None
DET errors	• `UART_E_PARAM_CONFIG`: Invalid pointer. • `UART_E_ALREADY_INITIALIZED`: Driver already initialized.
DEM errors	None
Description	This function initializes all local data for the configured channels. The driver state is set to IDLE.

4.4.2 Uart_DeInit

Table 39 Uart_DeInit

Syntax	`FUNC(void, UART_CODE) Uart_DeInit(void)`
Service ID	0x10
Sync/Async	Sync
Reentrancy	Non-reentrant
Parameters (in)	None
Parameters (out)	None
Return value	None
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_TRANSACTION`: Driver not in IDLE state.
DEM errors	None
Description	Initializes all local data and registers to their reset values. The driver state is set to UNINIT.

4.4.3 Uart_SetCommMode

Table 40 Uart_SetCommMode

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_SetCommMode`
	`(`
	`  CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId,`
	`  CONST(Uart_CommIdType, AUTOMATIC) CommId`
	`)`

Service ID	0x11
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `CommId`: Communication setting ID.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in IDLE state. • `UART_E_PARAM_COMMID`: Invalid communication setting ID.
DEM errors	None
Description	This operation changes the current communication settings. The UART driver uses the default communication settings for initialization. The default communication settings are specified with the configuration; these values are related to the MCU clock settings. However, the MCU clock settings can be changed by the MCU

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_SetCommMode`
	`(`
	`  CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId,`
	`  CONST(Uart_CommIdType, AUTOMATIC) CommId`
	`)`

Service ID	0x11
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `CommId`: Communication setting ID.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in IDLE state. • `UART_E_PARAM_COMMID`: Invalid communication setting ID.
DEM errors	None
	functionality at run time. In such cases, by using this service, you can continue to use the UART driver even if the SCB clock was changed. *Note:* *You must ensure proper configuration of the communication settings and select it. There is no check for coherency between the SCB clock and the communication settings. After reinitialization, the default settings will be used again.*

4.4.4 Uart_GetCommMode

Table 41 Uart_GetCommMode

Syntax	`FUNC(Uart_CommIdType, UART_CODE) Uart_GetCommMode ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId )`
Service ID	0x12
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	`Uart_CommIdType`: Communication setting ID.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_CHANNEL`: Invalid channel ID.
DEM errors	None
Description	This service returns the current communication setting ID.

4.4.5 Uart_GetTxStatus

Table 42 Uart_GetTxStatus

Syntax	`FUNC(Uart_ChannelTxStatusType, UART_CODE) Uart_GetTxStatus ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId )`
Service ID	0x37
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID
Parameters (out)	None
Return value	• `UART_UNKNOWN_STATUS_TX`: UART driver cannot judge the transmission status. • `UART_UNINIT_TX`: UART driver is uninitialized. • `UART_IDLE_TX`: UART driver is not operating in transmission. • `UART_BUSY_ASYNCTX`: UART driver is operating in asynchronous transmission. • `UART_BUSY_SYNCTX`: UART driver is operating in synchronous transmission. • `UART_ERRORTX`: UART driver detected a transmission error.
DET errors	`UART_E_PARAM_CHANNEL`: Invalid channel ID.
DEM errors	None
Description	This service returns the current transmission status.

4.4.6 Uart_GetRxStatus

Table 43 Uart_GetRxStatus

Syntax	`FUNC(Uart_ChannelRxStatusType, UART_CODE) Uart_GetRxStatus ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId )`
Service ID	0x38
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	• `UART_UNKNOWN_STATUS_RX`: UART driver cannot judge the reception status. • `UART_UNINIT_RX`: UART driver is uninitialized. • `UART_IDLE_RX`: UART driver is not operating in reception. • `UART_BUSY_AUTORX`: UART driver is operating in asynchronous reception without API. • `UART_BUSY_ASYNCRX`: UART driver is operating in asynchronous reception. • `UART_BUSY_SYNCRX`: UART driver is operating in synchronous reception. • `UART_ERRORRX`: UART driver detected a reception error.
DET errors	`UART_E_PARAM_CHANNEL`: Invalid channel ID.
DEM errors	None
Description	This service returns the current reception status.

4.4.7 Uart_SetupEb

Table 44 Uart_SetupEb

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_SetupEb ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId, P2VAR(Uart_BufferType, AUTOMATIC, UART_APPL_DATA) RingBufPtr, CONST(Uart_BufferSizeType, AUTOMATIC) RingBufSize )`
Service ID	0x3A
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `RingBufPtr`: Pointer to ring buffer start address. • `RingBufSize`: Size of ring buffer (number of bytes).
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_POINTER`: Invalid `RingBufPtr`. • `UART_E_PARAM_LENGTH`: Invalid `RingBufSize`. • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in IDLE state.
DEM errors	None
Description	This service saves the ring buffer information for subsequent reception. The saved ring buffer information will be used repeatedly in subsequent operations until this request is invoked again. The received data will be stored into this ring buffer. *Note:* `RingBufSize` *must be smaller than the configured maximum value (*`UartMaxBuffersize`*).* *Note:* `RingBufSize` *must be even number, if* `UartRxDataWidth` *is specified nine bits. The received data is stored into the ring buffer in halfword, even though the data width is specified nine bits.*

4.4.8 Uart_SyncTransmit

Table 45 Uart_SyncTransmit

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_SyncTransmit ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId, P2VAR(Uart_BufferType, AUTOMATIC, UART_APPL_DATA) TxBufPtr, CONST(Uart_BufferSizeType, AUTOMATIC) TxSize )`
Service ID	0x40
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `TxBufPtr`: Pointer to the data to transmit. • `TxSize`: Size of the transmit data (number of bytes).
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_POINTER`: Invalid `TxBufPtr`. • `UART_E_PARAM_LENGTH`: Invalid `TxSize`. • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in IDLE state, or a timeout was detected.
DEM errors	None
Description	This service starts the transmission operation. The provided data will be transmitted to the UART bus, and if the transmission is completed, this function returns the result. If the transmission continues longer than the configured value (`UartTxTimeout`), this function detects the timeout and returns the error code. *Note: The provided data is simply copied into the FIFO byte by byte (in case nine data bits, halfword by halfword). The UART driver does not take care of the number of data bits while copying the data to the FIFO. If you use less than eight data bits or nine data bits, you must handle the bit position of the data.* *Note: `TxSize` must be even number, if `UartTxDataWidth` is specified nine bits. The provided data is copied into the FIFO in halfword, even though the data width is specified nine bits.*

4.4.9 Uart_SyncReceive

Table 46 Uart_SyncReceive

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_SyncReceive ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId, P2VAR(Uart_BufferType, AUTOMATIC, UART_APPL_DATA) RxBufPtr, CONST(Uart_BufferSizeType, AUTOMATIC) RxSize )`
Service ID	0x3C
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `RxBufPtr`: Pointer to the reception buffer. • `RxSize`: Size of the data to receive (number of bytes).
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_POINTER`: Invalid `RxBufPtr`. • `UART_E_PARAM_LENGTH`: Invalid `RxSize`. • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in BUSY_AUTO state, or a timeout was detected.
DEM errors	None
Description	This service starts a synchronous reception operation. The received data will be stored into the provided buffer. If the specified length was stored, this function returns the result. If the reception continues longer than the configured value (`UartRxTimeout`), this function detects the timeout and returns the error code. *Note: The received data is simply copied from the FIFO byte by byte (in case nine data bits, halfword by halfword). The UART driver does not take care of the number of data bits while copying the data from the FIFO. If you use less than eight data bits or nine data bits, you must handle the bit position of the data.* *Note: `RxSize` must be even number, if `UartRxDataWidth` is specified nine bits. The received data is stored into the provided buffer in halfword, even though the data width is specified nine bits.*

4.4.10 Uart_AsyncTransmit

Table 47 Uart_AsyncTransmit

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_AsyncTransmit`
	`(`
	`  CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId,`
	`  P2VAR(Uart_BufferType, AUTOMATIC, UART_APPL_DATA) TxBufPtr,`
	`  CONST(Uart_BufferSizeType, AUTOMATIC) TxSize`
	`)`

Service ID	0x41
Sync/Async	Async
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `TxBufPtr`: Pointer to the data to transmit. • `TxSize`: Size of the transmit data (number of bytes).
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_POINTER`: Invalid `TxBufPtr`. • `UART_E_PARAM_LENGTH`: Invalid `TxSize`. • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in IDLE state.
DEM errors	None
Description	This service starts a transmission operation. The provided data will be transmitted to the UART bus. After this function call, the ISR continues the remaining transmit operation. *Note: The provided data is simply copied into the FIFO byte by byte (in case nine data bits, halfword by halfword). The UART driver does not take care of the number of data bits while copying the data to the FIFO. If you use less than eight data bits or nine data bits, you must handle the bit position of the data.* *Note: `TxSize` must be even number, if `UartTxDataWidth` is specified nine bits. The provided data is copied into the FIFO in halfword, even though the data width is specified nine bits.*

4.4.11 Uart_AsyncReceive

Table 48 Uart_AsyncReceive

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_AsyncReceive ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId, CONST(Uart_BufferSizeType, AUTOMATIC) RxSize )`
Service ID	0x3D
Sync/Async	Async
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `RxSize`: Size of the data to receive (number of bytes).
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_LENGTH`: Invalid `RxSize`. • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in BUSY_AUTO state.
DEM errors	None
Description	This service starts an asynchronous reception operation. The driver sets the specified receive size as a notification criterion. If data of the specified size is received, a notification function is called from the ISR. *Note: The received data is simply copied from the FIFO byte by byte (in case nine data bits, halfword by halfword). The UART driver does not take care of the number of data bits while copying the data to the FIFO. If you use less than eight data bits or nine data bits, you must handle the bit position of the data.* *Note: `RxSize` must be even number, if `UartRxDataWidth` is specified nine bits. The received data is stored into the ring buffer in halfword, even though the data width is specified nine bits.*

4.4.12 Uart_CancelTransmit

Table 49 Uart_CancelTransmit

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_CancelTransmit ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId )`
Service ID	0x43
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_CHANNEL`: Invalid channel ID.
DEM errors	None
Description	This service terminates the ongoing transmission operation. After termination, the transmit status is changed to IDLE.

Note: If you call Uart_CancelTransmit during the ongoing transaction, the receiving node might detect some error (for example, Parity error). Please note it when you are using.

4.4.13 Uart_CancelReceive

Table 50 Uart_CancelReceive

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_CancelReceive ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId )`
Service ID	0x3F
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_CHANNEL`: Invalid channel ID.
DEM errors	None
Description	This service terminates the ongoing reception operation. After termination, the receive status is changed to IDLE. This means that AUTO reception is stopped.

4.4.14 Uart_StartUartRx

Table 51 Uart_StartUartRx

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_StartUartRx`
	`(`
	`  CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId`
	`)`

Service ID	0x3B
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in IDLE state.
DEM errors	None
Description	This service restarts the AUTO reception operation. *Note: All previously received data will be ignored. This means that the ring buffer data will be overwritten.*

4.4.15 Uart_GetRingBufInfo

Table 52 Uart_GetRingBufInfo

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_GetRingBufInfo (CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId, P2VAR(P2VAR(Uart_BufferType, AUTOMATIC, UART_APPL_DATA), AUTOMATIC, UART_APPL_DATA) TopPtrPtr, P2VAR(Uart_BufferSizeType, AUTOMATIC, UART_APPL_DATA) RingBufSizePtr, P2VAR(P2VAR(Uart_BufferType, AUTOMATIC, UART_APPL_DATA), AUTOMATIC, UART_APPL_DATA) PreTailPtrPtr, P2VAR(Uart_BufferSizeType, AUTOMATIC, UART_APPL_DATA) RingBufEmptySizePtr)`
Service ID	0x3E
Sync/Async	Sync
Reentrancy	Reentrant
Parameters(in)	`ChannelId`: Channel ID.
Parameters(out)	• `TopPtrPtr`: Newest data position. • `RingBufSizePtr`: Total size of the ring buffer (number of bytes). • `PreTailPtrPtr`: Oldest data position (not yet read). • `RingBufEmptySizePtr`: Empty size of the ring buffer (number of bytes).
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver not initialized • `UART_E_PARAM_POINTER`: Invalid pointer (any output parameter). • `UART_E_PARAM_CHANNEL`: Invalid channel ID.
DEM errors	None
Description	This service provides the current ring buffer information. `TopPtrPtr` is changed by each data reception. `PreTailPtrPtr` is changed by each SyncReceive/AsyncReceive API call. `RingBufSizePtr` is changed by each SetUpEB call. `RingBufEmptySizePtr` is changed by each data reception.

4.4.16 Uart_GetVersionInfo

Table 53 Uart_GetVersionInfo

Syntax	`FUNC(void, UART_CODE) Uart_GetVersionInfo ( P2VAR(Std_VersionInfoType, AUTOMATIC, UART_APPL_DATA) VersionInfo )`
Service ID	0x39
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	`VersionInfo`: Version information.
Return value	None
DET errors	`UART_E_PARAM_POINTER`: Invalid pointer.
DEM errors	None
Description	This service provides the version information of this module. This includes module ID, vendor ID, and vendor-specific version numbers.

4.4.17 Uart_GetTxBufferInfo

Table 54 Uart_GetTxBufferInfo

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_GetTxBufferInfo ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId, P2VAR(P2VAR(Uart_BufferType, AUTOMATIC, UART_APPL_DATA), AUTOMATIC, UART_APPL_DATA) TopPtrPtr, P2VAR(Uart_BufferSizeType, AUTOMATIC, UART_APPL_DATA) TxBufSizePtr )`
Service ID	0x42
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	• `TopPtrPtr`: Position of the next data to transmit. • `TxBufSizePtr`: Size of the remaining data to transmit (number of bytes).
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_POINTER`: Invalid pointer (any output parameter). • `UART_E_PARAM_CHANNEL`: Invalid channel ID.
DEM errors	None
Description	This service provides the current transmit buffer information. `TopPtrPtr` and `TxBufSizePtr` are changed by each data transmission.

4.4.18 Uart_GetTxIntStatus

Table 55 Uart_GetTxIntStatus

Syntax	`FUNC(Uart_TxIntStatusType, UART_CODE) Uart_GetTxIntStatus ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId )`
Service ID	0x45
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	• `UART_TX_INT_IDLE`: UART driver is not operating in transmission. • `UART_TX_INT_BUSY_ASYNCTX`: UART driver is operating in asynchronous transmission. • `UART_TX_INT_ERRORTX`: UART driver detected a transmission error. • `UART_TX_INT_UNKNOWN_STATUS_TX`: UART driver cannot judge the transmission status.
DET errors	• `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_PARAM_POINTER`: Invalid pointer (any member of configuration structure).
DEM errors	• `UART_E_HW_TX_OVERFLOW_ERROR`: Tx FIFO overflow. • `UART_E_HW_DW_SRC_BUS_ERROR`: DW source bus error. • `UART_E_HW_DW_DST_BUS_ERROR`: DW destination bus error. • `UART_E_HW_DW_SRC_MISAL`: DW source address misalignment. • `UART_E_HW_DW_DST_MISAL`: DW destination address misalignment. • `UART_E_DW_HW_CURR_PTR_NULL`: DW current descriptor pointer is null. • `UART_E_DW_HW_ACTIVE_CH_DIABLED`: DW active channel is disabled. • `UART_E_DW_HW_DESCR_BUS_ERROR`: DW descriptor bus error.
Description	This service returns the current transmission status by reading the interrupt status of the selected UART Tx channel and DW channel. *Note: This function is called from VUART driver. Please do not call this API in your application.*

4.4.19 Uart_AsyncTransmitWithHeader

Table 56 Uart_AsyncTransmitWithHeader

Syntax	`FUNC(Std_ReturnType, UART_CODE) Uart_AsyncTransmitWithHeader ( CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId, P2CONST(Uart_HeaderType, AUTOMATIC, UART_APPL_CONST) HeaderPtr, P2CONST(Uart_BufferType, AUTOMATIC, UART_APPL_CONST) TxBufPtr )`
Service ID	0x44
Sync/Async	Async
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `HeaderPtr`: Pointer to the header data to transmit . • `TxBufPtr`: Pointer to the data to transmit.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `UART_E_UNINIT`: Driver uninitialized. • `UART_E_PARAM_POINTER`: Invalid pointer (input parameter or member of configuration structure). • `UART_E_PARAM_CHANNEL`: Invalid channel ID. • `UART_E_TRANSACTION`: Driver/channel not in IDLE state.
DEM errors	None
Description	This service starts a transmission operation for pixel data. The provided header and data will be transmitted to the UART bus. After this function call, the DW continues the remaining transmit operation. *Note: This function is called from VUART driver. Please do not call this API in your application.*

4.5 Scheduled functions

None

4.6 Interrupt service routine

4.6.1 Uart_Interrupt_SCB<n>_CatX

Table 57 Uart_Interrupt_SCB<n>_CatX

Syntax	`ISR_NATIVE(Uart_Interrupt_SCB<n>_Cat1) or ISR(Uart_Interrupt_SCB<n>_Cat2)`
Service ID	0x5B
Sync/Async	Sync
Reentrancy	Non-reentrant
Parameters (in)	None
Parameters (out)	None
Return value	None
DET errors	None
DEM errors	• `UART_E_RINGBUFER_OVERFLOW`: Ring buffer has been overwritten without read. • `UART_E_HW_RX_PARITY_ERROR`: Parity error detected. • `UART_E_HW_RX_FRAME_ERROR`: Frame error detected. • `UART_E_HW_TX_OVERFLOW_ERROR`: Tx FIFO overflow. • `UART_E_HW_RX_OVERFLOW_ERROR`: Rx FIFO overflow. `UART_E_HW_RX_UNDERFLOW_ERROR`: Rx FIFO underflow.
Description	This function is an ISR. This function progresses the transmit/receive operation of the affected channels. If the specified transmit/receive length is reached, the corresponding notification function is called.

4.7 Required callback functions

4.7.1 UART notification function

The UART driver uses the following callback routines to inform other software modules about certain states or state changes.

All notification functions must be reentrant. UART driver API calls are not allowed from callback functions.

4.7.1.1 Uart_TxNotification

Table 58 Uart_TxNotification

Syntax	`void My_TxNotification`
	`(`
	`  CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId`
	`)`

Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	None
Description	This notification is a user-provided callback routine to notify that transmission has been completed. The name of notification function is specified by `UartTxNotificationCalloutFunction`.

4.7.1.2 Uart_RxNotification

Table 59 Uart_RxNotification

Syntax	`void My_RxNotification`
	`(`
	`  CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId`
	`)`

Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	None
Description	This notification is a user-provided callback routine to notify that reception has been completed. The name of notification function is specified by `UartRxNotificationCalloutFunction`.

4.7.1.3 Uart_TxErrorNotification

Table 60 Uart_TxErrorNotification

Syntax	`void My_TxErrorNotification`
	`(`
	`  CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId`
	`)`

Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	None
Description	This notification is a user-provided callback routine to notify that transmission has been completed with an error. The name of notification function is specified by `UartTxErrorNotificationCalloutFunction`.

4.7.1.4 Uart_RxErrorNotification

Table 61 Uart_RxErrorNotification

Syntax	`void My_RxErrorNotification`
	`(`
	`  CONST(Uart_ChannelIdType, AUTOMATIC) ChannelId`
	`)`

Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	None
Description	This notification is a user-provided callback routine to notify that reception has been completed with an error. The name of notification function is specified by `UartRxErrorNotificationCalloutFunction`.

4.7.2 DET

If default error detection is enabled, the UART driver uses the following callback function provided by DET. If you do not use DET, you must implement this function within your application.

4.7.2.1 Det_ReportError

Table 62 Det_ReportError

Syntax	`Std_ReturnType Det_ReportError`
	`(`
	`  uint16 ModuleId,`
	`  uint8 InstanceId,`
	`  uint8 ApiId,`
	`  uint8 ErrorId`
	`)`

Parameters (in)	• `ModuleId`: Module ID of the calling module. • `InstanceId`: Instance ID of the calling module. • `ApiId`: ID of the API service that calls this function. • `ErrorId`: ID of the detected development error.
Parameters (out)	None
Return value	Always returns `E_OK`.
Description	Service for reporting development errors.

4.7.3 DEM

If DEM notifications are enabled, the UART driver uses the following callback function provided by DEM. If you do not use DEM, you must implement this function within your application.

4.7.3.1 Dem_ReportErrorStatus

Table 63 Dem_ReportErrorStatus

Syntax	`void Dem_ReportErrorStatus`
	`(`
	`  Dem_EventIdType EventId,`
	`  Dem_EventStatusType EventStatus`
	`)`

Parameters (in)	• `EventId`: Identification of an event by the assigned event ID. • `EventStatus`: Test result of the given event.
Parameters (out)	None
Return value	None
Description	Service for reporting diagnostic events.

4.7.4 Error Callout functions

The UART driver requires an error callout handler. Every error is reported to this handler; error checking cannot be switched off.

4.7.4.1 Error Callout function

Table 64 Error Callout function

Syntax	`void My_ErrorCalloutHandler`
	`(`
	`  uint16 ModuleId,`
	`  uint8 InstanceId,`
	`  uint8 ApiId,`
	`  uint8 ErrorId`
	`)`

Parameters (in)	• `ModuleId`: Module ID of the calling module. • `InstanceId`: Instance ID of the calling module. • `ApiId`: ID of the API service that calls this function. • `ErrorId`: ID of the detected error.
Parameters (out)	None
Return value	None
Description	Service for reporting errors. The name of error callout function is specified by `UartErrorCalloutFunction`.

5 Appendix B: VUART API and external definitions

5.1 Include files

The *Vuart.h* file is the only file that needs to be included to use the functions of the VUART driver.

5.2 Data types

5.2.1 Vuart_ChannelIdType

Table 65 Vuart_ChannelIdType

Type	uint8
Description	Type of channel ID. This type is used to specify a channel.

5.2.2 Vuart_BufferType

Table 66 Vuart_BufferType

Type	uint8
Description	Type of buffer element. This type is used for buffer reference pointers.

5.2.3 Vuart_HeaderType

Table 67 Vuart_HeaderType

Type	uint8
Description	Type of header buffer element. This type is used for header buffer reference pointers.

5.3 Constants

5.3.1 Error codes

A service may return one of the following error codes if default error detection is enabled.

Table 68 Error codes

Name	Value	Description
VUART_E_ALREADY_STARTED	0x11	VUART is already started.
VUART_E_PARAM_POINTER	0x12	Pointer out of range.
VUART_E_PARAM_CHANNEL	0x14	Channel ID out of range.
VUART_E_TRANSACTION	0x15	Not called in IDLE status.
VUART_E_NOTSTARTED	0x16	VUART is not started.

5.3.2 Version / module information

Table 69 Version / module information

Name	Value	Description
VUART_SW_MAJOR_VERSION	See release notes	Vendor-specific major version number
VUART_SW_MINOR_VERSION	See release notes	Vendor-specific minor version number
VUART_SW_PATCH_VERSION	See release notes	Vendor-specific patch version number
VUART_MODULE_ID	255	Module ID
VUART_VENDOR_ID	66	Vendor ID

5.3.3 API service ID

The following API service IDs are used when reporting errors via DET or via the error callout function.

Table 70 API service ID

Name	Value	API name
VUART_API_START	0x65	Vuart_Start
VUART_API_STOP	0x66	Vuart_Stop
VUART_API_UPDATE_BUFF	0x67	Vuart_UpdateBuffer
VUART_API_GET_VERSION_INFO	0x68	Vuart_GetVersionInfo
VUART_API_GET_CURRENT_ROW	0x69	Vuart_GetCurrentRow
VUART_API_MAIN_FUNC_CH[n]	0x6E+(n)	Vuart_MainFunction_Ch[n]

Note: `Ch[n]` is the number of the VuartChannelId.

5.4 Functions

5.4.1 Vuart_GetVersionInfo

Table 71 Vuart_GetVersionInfo

Syntax	`FUNC (void, VUART_CODE) Vuart_GetVersionInfo ( P2VAR(Std_VersionInfoType, AUTOMATIC, VUART_APPL_DATA) VersionInfoPtr )`
Service ID	0x68
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	`VersionInfoPtr`: Version information.
Return value	None
DET errors	`VUART_E_PARAM_POINTER`: Invalid pointer.
DEM errors	None
Description	This service provides the version information of this module. This includes module ID, vendor ID, and vendor specific version numbers.

5.4.2 Vuart_Start

Table 72 Vuart_Start

Syntax	`FUNC(Std_ReturnType, VUART_CODE) Vuart_Start ( CONST(Vuart_ChannelIdType, AUTOMATIC) ChannelId )`
Service ID	0x65
Sync/Async	Sync
Reentrancy	Non-reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `VUART_E_PARAM_CHANNEL`: Invalid channel ID. • `VUART_E_ALREADY_STARTED`: VUART is already started.
DEM errors	None
Description	This service prepares the transmission operation. To enable the transmission operation, this function calls `Uart_Init`, if the UART driver is not initialized.

5.4.3 Vuart_Stop

Table 73 Vuart_Stop

Syntax	`FUNC(Std_ReturnType, VUART_CODE) Vuart_Stop`
	`(`
	`  CONST(Vuart_ChannelIdType, AUTOMATIC) ChannelId`
	`)`

Service ID	0x66
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK` Request declined.
DET errors	• `VUART_E_PARAM_CHANNEL`: Invalid channel ID. • `VUART_E_NOTSTARTED`: VUART is not started.
DEM errors	None
Description	This service stops the transmission operation. To stop the transmission operation, this function calls `Uart_CancelTransmit` regardless of ongoing transmission or not, if the UART driver is already initialized. *Note: While `Vuart_MainFunction_Ch[n]` is running, this service does not stop the transmission operation.*

5.4.4 Vuart_UpdateBuffer

Table 74 Vuart_UpdateBuffer

Syntax	`FUNC(Std_ReturnType, VUART_CODE) Vuart_UpdateBuffer ( CONST(Vuart_ChannelIdType, UART_CONST) ChannelId, P2CONST(Vuart_BufferType, AUTOMATIC, VUART_APPL_CONST) BufferPtr )`
Service ID	0x67
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID. • `BufferPtr`: Pointer to the image frame data to transmit.
Parameters (out)	None
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `VUART_E_PARAM_CHANNEL`: Invalid channel ID. • `VUART_E_PARAM_POINTER`: Invalid `BufferPtr`.
DEM errors	None
Description	This service saves the buffer pointer of the image frame data for subsequent transmission. The VUART driver can keep only the buffer pointer for currently ongoing transmission (current buffer) and the buffer pointer for subsequent transmission (next buffer). Once the transmission completed of current buffer, the VUART driver replaces the current buffer pointer by the next buffer pointer. The saved buffer pointer will be used repeatedly in subsequent operations until this request is invoked again. *Note: The provided data is not copied into another area by the VUART driver. The user should prepare at least double buffer to avoid data corruption for ongoing transmission.*

5.4.5 Vuart_GetCurrentRow

Table 75 Vuart_GetCurrentRow

Syntax	`FUNC(Std_ReturnType, VUART_CODE) Vuart_GetCurrentRow ( CONST(Vuart_ChannelIdType, UART_CONST) ChannelId, P2VAR(uint16, AUTOMATIC, VUART_APPL_DATA) CurrentRowPtr )`
Service ID	0x69
Sync/Async	Sync
Reentrancy	Reentrant
Parameters (in)	• `ChannelId`: Channel ID.
Parameters (out)	• `CurrentRowPtr`: Current row number.
Return value	• `E_OK`: Request accepted. • `E_NOT_OK`: Request declined.
DET errors	• `VUART_E_PARAM_CHANNEL`: Invalid channel ID. • `VUART_E_PARAM_POINTER`: Invalid `CurrentRowPtr`.
DEM errors	None
Description	This service provides the current row number which is currently being transferred or the blank row number currently being proceeded. If the blank row number is being processed, "transferred row number + current blank row number" is the output. `CurrentRowPtr` is changed by each `Vuart_MainFunction_Ch[n]` call.

5.5 Scheduled functions

5.5.1 Vuart_MainFunction_Ch[n]

Table 76 Vuart_MainFunction_Ch[n]

Syntax	`FUNC(void, VUART_CODE) Vuart_MainFunction_Ch[n](void)`
Service ID	0x6E + (n)
Sync/Async	Async
Reentrancy	Non-reentrant
Parameters (in)	None
Parameters (out)	None
Return value	None
DET errors	• `VUART_E_TRANSACTION`: Not called in IDLE status.
DEM errors	None
Description	This scheduled function shall be called periodically in order to transfer the payload for each row (including blank row). *Note: This function shall be called periodically in sufficient duration to transfer 1 row payload data.*

Note: `Ch[n]` is the number of the VuartChannelId.

5.6 Interrupt service routine

None

5.7 Required callback functions

5.7.1 VUART notification function

The VUART driver uses the following callback routines to inform other software modules about certain states or state changes.

All notification functions must be reentrant. VUART driver API calls are not allowed from callback functions.

5.7.1.1 Vuart_SwapCompleted

Table 77 Vuart_SwapCompleted

Syntax	`FUNC(void, UART_CODE) Vuart_SwapCompleted`
	`(`
	`  VAR(uint8, AUTOMATIC) ChannelId`
	`)`

Parameters (in)	`ChannelId`: Channel ID.
Parameters (out)	None
Return value	None
Description	This notification is a user provided callback routine to notify that all image frame data transmission completed for current buffer (including blank row) required by application. The name of notification function is specified by `VuartSwapNotificationCalloutFunction`.

5.7.2 DET

If default error detection is enabled, the VUART driver uses the following callback function provided by DET. If you do not use DET, you must implement this function within your application.

5.7.2.1 Det_ReportError

Table 78 Det_ReportError

Syntax	`Std_ReturnType Det_ReportError ( uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId )`
Parameters (in)	• `ModuleId`: Module ID of the calling module. • `InstanceId`: Instance ID of the calling module. • `ApiId`: ID of the API service that calls this function. • `ErrorId`: ID of the detected development error.
Parameters (out)	None
Return value	Always returns `E_OK`.
Description	Service for reporting development errors.

5.7.3 DEM

None

5.7.4 Error Callout functions

The VUART driver requires an error callout handler. Every error is reported to this handler; error checking cannot be switched off.

Table 79 Error Callout function

Syntax	`void My_ErrorCalloutHandler`
	`(`
	`  uint16 ModuleId,`
	`  uint8 InstanceId,`
	`  uint8 ApiId,`
	`  uint8 ErrorId`
	`)`

Parameters (in)	• `ModuleId`: Module ID of the calling module. • `InstanceId`: Instance ID of the calling module. • `ApiId`: ID of the API service that calls this function. • `ErrorId`: ID of the detected error.
Parameters (out)	None
Return value	None
Description	Service for reporting errors. The name of error callout function is specified by `VuartErrorCalloutFunction`.

6 Appendix C: Registers

6.1 Access register table

Table 80 SCB access register table

Register	Bit	Access size	Value	Description	Timing	Mask value	Monitoring value
CTRL	31:0	Word (32 bits)	0x8200*00* *=depends on configuration	Initialize the register	After calling `Uart_Init`	0x9303D70F	0x8200*00* *=depends on configuration
			0x0300400F	Deinitialize the register	After calling `Uart_DeInit`	0x9303D70F	0x0300400F
UART_CTRL	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Uart_Init`	0x03010000	0x00000000
			0x03000000	Deinitialize the register	After calling `Uart_DeInit`	0x03010000	0x03000000
UART_TX_CTRL	31:0	Word (32 bits)	0x000000** *=depends on configuration	Initialize the register	After calling `Uart_Init`	0x00000137	0x000000** *=depends on configuration
			0x00000002	Deinitialize the register	After calling `Uart_DeInit`	0x00000137	0x00000002
UART_RX_CTRL	31:0	Word (32 bits)	0x0*0*0*** *=depends on configuration	Initialize the register	After calling `Uart_Init`	0x010F3777	0x0*0*0*** *=depends on configuration
			0x000A0002	Deinitialize the register	After calling `Uart_DeInit`	0x010F3777	0x000A0002
UART_FLOW_CTRL	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Uart_Init`	0x0301007F	0x00000000
			0x00000000	Deinitialize the register	After calling `Uart_DeInit`	0x0301007F	0x00000000

Register	Bit	Access size	Value	Description	Timing	Mask value	Monitoring value
TX_CTRL	31:0	Word (32 bits)	0x000*0*0* *=depends on configuration	Initialize the register	After calling `Uart_Init`	0x0001011F	0x000*0*0* *=depends on configuration
			0x00000107	Deinitialize the register	After calling `Uart_DeInit`	0x0001011F	0x00000107
TX_FIFO_CTRL	31:0	Word (32 bits)	0x000000** *=depends on configuration	Initialize the register	After calling `Uart_Init`	0x000300FF	0x000000** *=depends on configuration
			0x00000000	Deinitialize the register	After calling `Uart_DeInit`	0x000300FF	0x00000000
TX_FIFO_STATUS	31:0	Word (32 bits)	-	Read-only register	During Tx transaction	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
TX_FIFO_WR	31:0	Word (32 bits)	Tx data	Tx data	During Tx transaction	-	Write-only register
RX_CTRL	31:0	Word (32 bits)	0x00000*0* *=depends on configuration	Initialize the register	After calling `Uart_Init`	0x0000031F	0x00000*0* *=depends on configuration
			0x00000107	Deinitialize the register	After calling `Uart_DeInit`	0x0000031F	0x00000107
RX_FIFO_CTRL	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Uart_Init`	0x000300FF	0x00000000
			0x00000000	Deinitialize the register	After calling `Uart_DeInit`	0x000300FF	0x00000000
RX_FIFO_STATUS	31:0	Word (32 bits)	-	Read-only register	During Rx transaction	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
RX_MATCH	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Uart_Init`	0x00FF00FF	0x00000000
			0x00000000	Deinitialize the register	After calling `Uart_DeInit`	0x00FF00FF	0x00000000

Register	Bit	Access size	Value	Description	Timing	Mask value	Monitoring value
RX_FIFO_RD	31:0	Word (32 bits)	-	Read-only register	When reading the received data	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
INTR_CAUSE	31:0	Word (32 bits)	0x0000000* *=depends on interrupt cause (Read-only)	Interrupt cause (Read-only)	During Rx/Tx transaction (interrupt)	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
INTR_TX	31:0	Word (32 bits)	0x000007F3	Initialize the register	After calling `Uart_Init`	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
			0x000007F3	Deinitialize the register	After calling `Uart_DeInit`	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
			0x00000*** *=depends on interrupt cause	Tx interrupt cause (read and clear with write)	During transmission (interrupt) After calling `Vuart_MainFunction_Ch[n]/ Vuart_Stop`	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
INTR_TX_MASK	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Uart_Init`	0x000007F3	0x00000000
			0x00000000	Deinitialize the register	After calling `Uart_DeInit`	0x000007F3	0x00000000
			0x00000000	Clear the interrupt mask	After runtime error detection	0x000007F3	0x00000000
			0x00000000	Clear the interrupt mask	After calling `Uart_CancelTransmit`	0x000007F3	0x00000000
			0x00000221	Set the interrupt mask	After calling `Uart_SyncTransmit/ Uart_AsyncTransmit`	0x000007F3	0x00000221
			0x00000200	Set the interrupt mask only for `UART_DONE`. Clear other interrupt masks	TRIGGER interrupt request generated and transmission completed	0x000007F3	0x00000200

Register	Bit	Access size	Value	Description	Timing	Mask value	Monitoring value
			0x00000000	Clear the interrupt mask.	UART_DONE interrupt request generated and transmission completed	0x000007F3	0x00000000
INTR_TX_MASKED	31:0	Word (32 bits)	-	Read-only register	During Tx transaction	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
INTR_RX	31:0	Word (32 bits)	0x00000FED	Initialize the register	After calling `Uart_Init`	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
			0x00000FED	Initialize the register	After calling `Uart_DeInit`	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
			0x00000*** *=depends on interrupt cause	Rx interrupt cause (read and clear with write)	During Rx transaction (interrupt)	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)
INTR_RX_MASK	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Uart_Init`	0x00000FED	0x00000000
			0x00000000	Deinitialize the register	After calling `Uart_DeInit`	0x00000FED	0x00000000
			0x00000000	Clear the interrupt mask	After runtime error detection	0x00000FED	0x00000000
			0x00000000	Clear the interrupt mask	After calling `Uart_CancelReceive`	0x00000FED	0x00000000
			0x0000036D	Set the interrupt mask	After the start of reception	0x00000FED	0x0000036D
INTR_RX_MASKED	31:0	Word (32 bits)	-	Read-only register	During Rx transaction	0x00000000 (Monitoring not needed.)	0x00000000 (Monitoring not needed.)

Note: *`Ch[n]` is the number of the VuartChannelId.*

Table 81 DW access register table

Register	Bit	Access size	Value	Description	Timing	Mask value	Monitoring value
CRC_CTL	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Vuart_MainFunction_Ch[n]`	0x00000101	0x00000000
			0x00000000	Deinitialize the register	After calling `Vuart_Stop`	0x00000101	0x00000000
CRC_DATA_CTL	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Vuart_MainFunction_Ch[n]`	0x000000FF	0x00000000
			0x00000000	Deinitialize the register	After calling `Vuart_Stop`	0x000000FF	0x00000000
CRC_POL_CTL	31:0	Word (32 bits)	0x******** *=depends on configuration	Initialize the register	After calling `Vuart_MainFunction_Ch[n]`	0xFFFFFFFF	0x******** *=depends on configuration
			0x00000000	Deinitialize the register	After calling `Vuart_Stop`	0xFFFFFFFF	0x00000000
CRC_REM_CTL	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Vuart_MainFunction_Ch[n]`	0xFFFFFFFF	0x00000000
			0x00000000	Deinitialize the register	After calling `Vuart_Stop`	0xFFFFFFFF	0x00000000
CRC_LFSR_CTL	31:0	Word (32 bits)	0x******** *=depends on configuration	Initialize the register	After calling `Vuart_MainFunction_Ch[n]`	0xFFFFFFFF	0x******** *=depends on configuration
			0x00000000	Deinitialize the register	After calling `Vuart_Stop`	0xFFFFFFFF	0x00000000
CH_CTL	31:0	Word (32 bits)	0x00000002	Initialize the register	After calling `Uart_Init`	0x80000BF7	0x00000002

Register	Bit	Access size	Value	Description	Timing	Mask value	Monitoring value
			0x*00000* *=depends on configuration	Channel control	After calling `Vuart_MainFunction_Ch[n]`	0x80000BF7	0x*00000* *=depends on configuration
			0x00000002	Deinitialize the register	After calling `Vuart_Stop`	0x80000BF7	0x00000002
CH_STATUS	31:0	Word (32 bits)	0x0000000* *=depends on interrupt cause	Channel status	During Tx transaction	0x8000000F	0x*000000* *=depends on interrupt cause
CH_IDX	31:0	Word (32 bits)	0x00000000	Initialize the register	After calling `Vuart_MainFunction_Ch[n]`	0x0000FFFF	0x0000**** *=depends on configuration
CH_CURR_PTR	31:0	Word (32 bits)	0x******** *=depends on configuration	Channel current descriptor pointer	After calling `Vuart_MainFunction_Ch[n]`	0xFFFFFFFC	0x******** *=depends on configuration
INTR	31:0	Word (32 bits)	0x00000001	Clear Interrupt	After calling `Vuart_MainFunction_Ch[n]`	0x00000001	0x00000000
			0x00000001	Deinitialize the register	After calling `Vuart_Stop`	0x00000001	0x00000000

Note: `Ch[n]` *is the number of the VuartChannelId.*

Note: *DW access only when VUART is enabled.*

Revision history

Version	Date	Description
**	2021-02-15	Initial version
*A	2021-06-11	Added note to improve the description in **Sleep mode** and **Uart_CancelTransmit**
*B	2021-08-18	Added a note in 6.3 Interrupts
*C	2021-12-21	Updated to the latest branding guidelines.
*D	2022-06-28	Removed unnecessary description “invalid interrupt cause detection” in **Table 8 General configuration**. Removed unnecessary definition UART_E_HW_INVALID_INTERRUPT_ERROR in **Table 14 Production error list** and **Table 35 Error codes** and **Table 57 Uart_Interrupt_SCB<n>_CatX**.
*E	2022-09-20	Added **VUART driver** and **Appendix B: VUART API and external definitions** for VUART driver. Added description about VUART driver in **General overview** and **UART driver** and **Appendix A: UART API and external definitions**. Added DW registers in **Access register table**.
*F	2023-01-24	Fixed typo of configuration parameter name to UartTxNotificationCalloutFunction, UartRxNotificationCalloutFunction, UartTxErrorNotificationCalloutFunction, UartRxErrorNotificationCalloutFunction. Updated the range of configuration parameter below. • UartTxTriggerLevel • UartTxDataWidth • UartRxDataWidth • UartDWCRCPolynomial • UartDWCRCSeed Added explanation about nine data bits in **UART driver functionality and operation** and **Functions** for UART driver. Updated explanation in **Functions** and **Required callback functions** for UART driver. Updated description of VuartHeaderTableData in **VuartConfigSet**. Updated state machine and description in **State transition**. Added **Call Vuart_GetCurrentRow** and **Vuart_GetCurrentRow** for Vuart_GetCurrentRow. Added definition VUART_API_GET_CURRENT_ROW in **API service ID**. Removed unnecessary definition UART_E_HW_TX_UNDERFLOW_ERROR in **Error codes** and **Uart_Interrupt_SCB<n>_CatX**.

Revision history

Version	Date	Description
		Updated explanation in **Vuart_Stop** and **Required callback functions** and **EB tresos Studio configuration interface** for VUART driver. Removed unnecessary DET error VUART_E_NOTSTARTED in **Vuart_MainFunction_Ch[n]**. Updated value of registers in **Access register table**.
*G	2023-12-08	Web release. No content updates.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-12-08

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-32365 Rev. *G

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.