MICROSAR VStdLib

Technical Reference

Generic implementation of the Vector Standard Library

Version 1.00.00

Authors	Torsten Kercher
Status	Released

Document Information

History

Author	Date	Version	Remarks
Torsten Kercher	2015-05-04	1.00.00	Creation

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	1.6.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0

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

Contents

1 Component History 5

2 Introduction 6
2.1 Architecture Overview 6

3 Functional Description 8
3.1 Features 8
3.2 Initialization and Main Functions 8
3.3 Error Handling 8

4 Integration 9
4.1 Scope of Delivery 9
4.2 Include Structure 9
4.3 Critical Sections 9
4.4 Compiler Abstraction and Memory Mapping 10
4.5 Integration Hints 11

5 API Description 12
5.1 Type Definitions 12
5.2 Services provided by VStdLib 12
5.3 Services used by VStdLib 22

6 Configuration 23
6.1 Configuration Variants 23
6.2 Manual Configuration in Header File 23

7 Abbreviations 25

8 Contact 26

Illustrations

Figure 2-1 AUTOSAR 4.x Architecture Overview 6
Figure 2-2 Interfaces to adjacent modules 7
Figure 4-1 Include Structure 9

Tables

Table 1-1 Component history 5
Table 3-1 Service IDs 8
Table 3-2 Errors reported to DET 8
Table 4-1 Static files 9
Table 4-2 Compiler Abstraction and Memory Mapping 10
Table 5-1 VStdLib_GetVersionInfo 12
Table 5-2 VStdLib_MemClr 13
Table 5-3 VStdLib_MemClrMacro 14
Table 5-4 VStdLib_MemSet 15
Table 5-5 VStdLib_MemSetMacro 16
Table 5-6 VStdLib_MemCpy 17
Table 5-7 VStdLib_MemCpy16 18
Table 5-8 VStdLib_MemCpy32 19
Table 5-9 VStdLib_MemCpy_s 20
Table 5-10 VStdLib_MemCpyMacro 21
Table 5-11 VStdLib_MemCpyMacro_s 22
Table 5-12 Services used by VStdLib 22
Table 6-1 General configuration 24
Table 7-1 Abbreviations 25

1 Component History

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

Component Version	New Features
1.00	Creation of the component.
2.00	Detach the component from core-based implementations, give optimized routines and support operations on large data (> 65535 bytes).

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the generic Vector Standard Library (VStdLib).

Supported AUTOSAR Release*:	4.x	
Supported Configuration Variants:	pre-compile	
Vendor ID:	VSTDLIB_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	VSTDLIB_MODULE_ID	255 decimal (according to [1])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

The VStdLib provides a hardware independent implementation of memory manipulation services used by several MICROSAR BSW components.

2.1 Architecture Overview

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

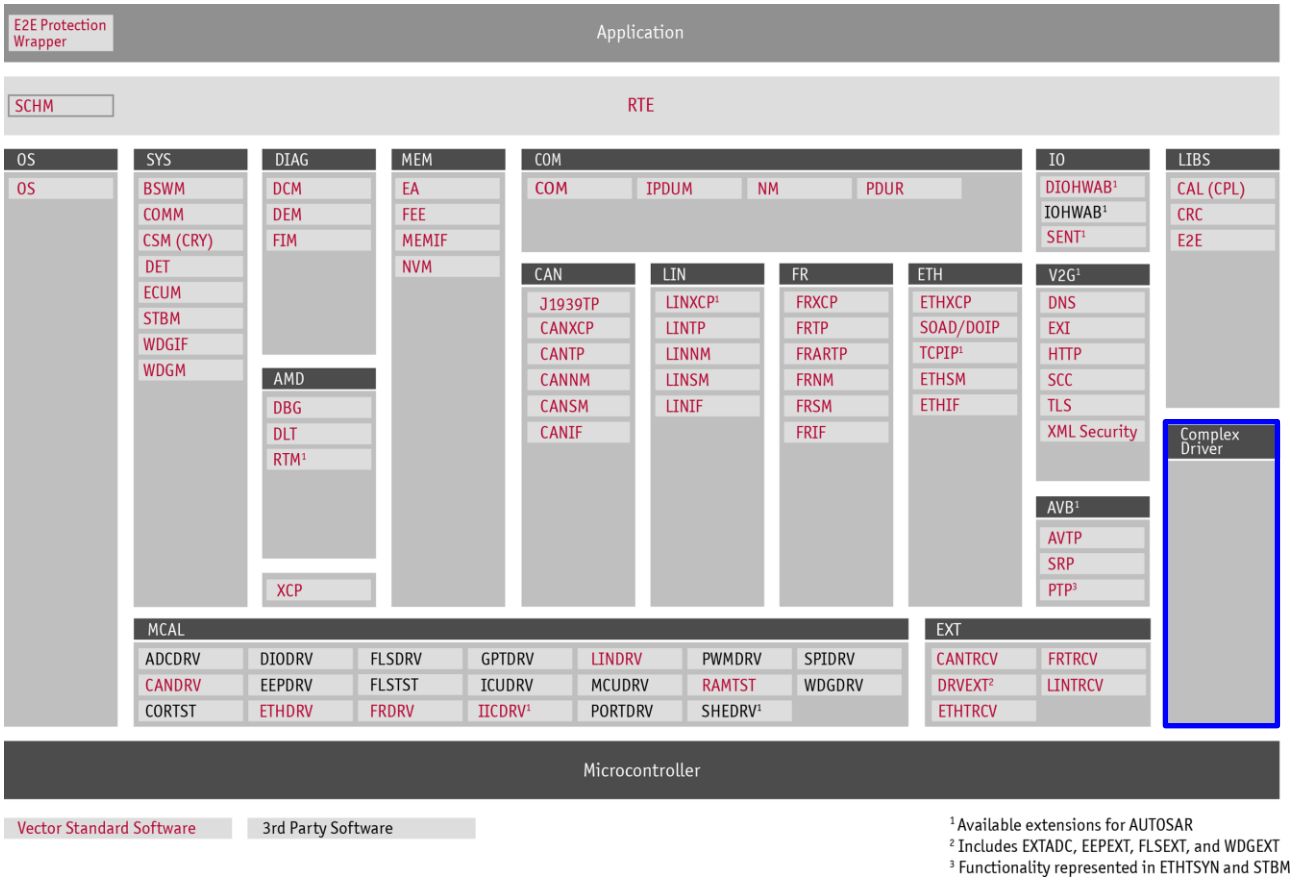

Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the VStdLib. These interfaces are described in chapter 5.

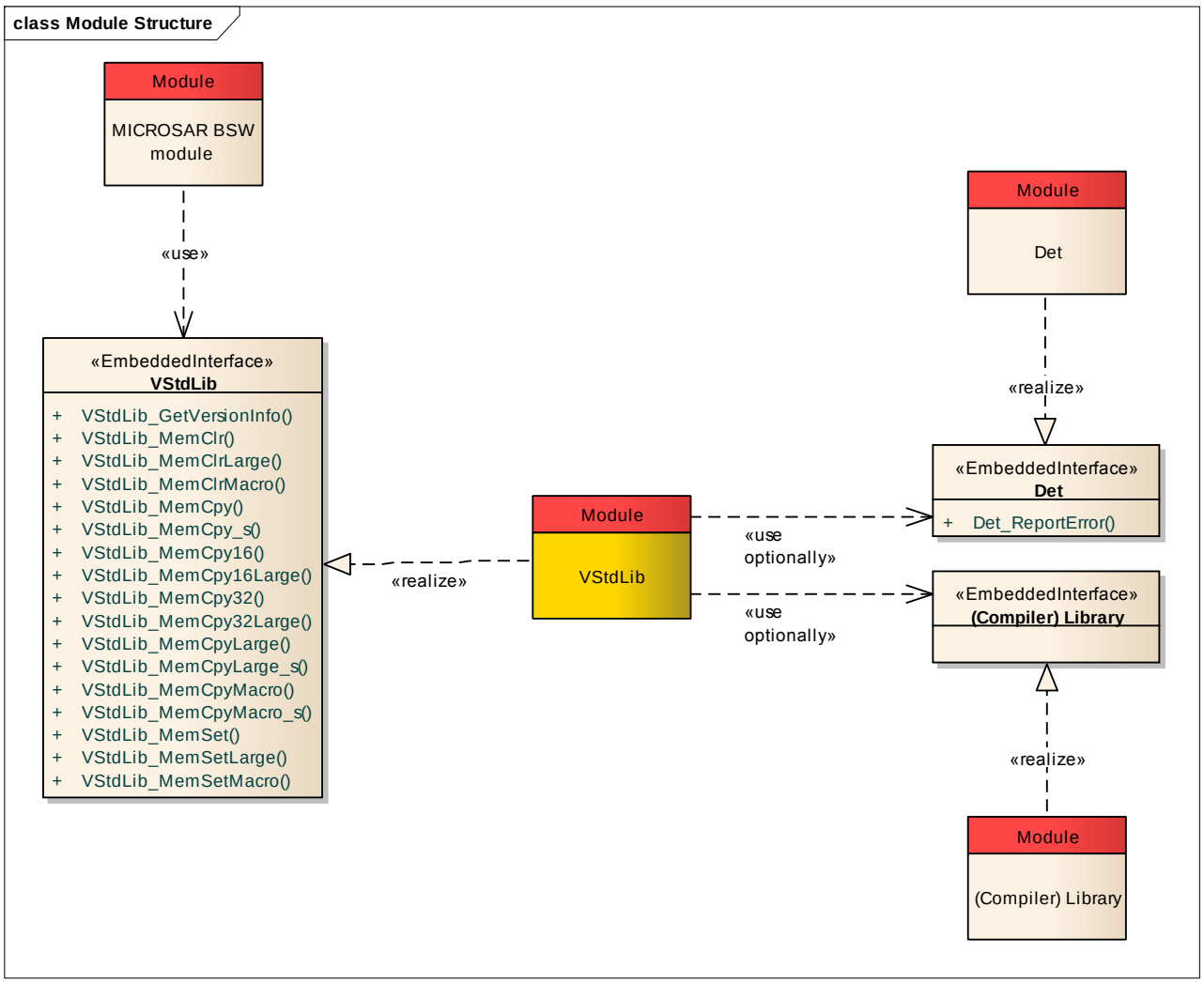

Figure 2-2 Interfaces to adjacent modules

3 Functional Description

This chapter describes the general function of the component.

3.1 Features

The Vector Standard Library gives a standard interface for memory initialization and copy services as described in section 5.2. It provides a hardware independent implementation of this interface, but also allows the mapping to project specific implementations for optimization reasons.

3.2 Initialization and Main Functions

No initialization is necessary and no main functions are provided.

3.3 Error Handling

3.3.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `VSTDLIB_DEV_ERROR_REPORT == STD_ON`).

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

The reported VStdLib ID is 255. The service IDs identify the services which are described in section 5.2. The following table presents the service IDs and the related services:

Service ID	Service
`VSTDLIB_SID_MEM_SET` (0x00)	`VStdLib_MemClr()`, `VStdLib_MemSet()`
`VSTDLIB_SID_MEM_COPY` (0x01)	`VStdLib_MemCpy()`
`VSTDLIB_SID_MEM_COPY_16` (0x02)	`VStdLib_MemCpy16()`
`VSTDLIB_SID_MEM_COPY_32` (0x03)	`VStdLib_MemCpy32()`
`VSTDLIB_SID_MEM_COPY_S` (0x04)	`VStdLib_MemCpy_s()`
`VSTDLIB_SID_GET_VERSION_INFO` (0x05)	`VStdLib_GetVersionInfo()`

Table 3-1 Service IDs

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

Error Code	Description
`VSTDLIB_E_PARAM_POINTER` (0x01)	API service used with NULL pointer parameter.
`VSTDLIB_E_PARAM_SIZE` (0x02)	`VStdLib_MemCpy_s()` used with invalid destination size parameter.

Table 3-2 Errors reported to DET

3.3.2 Production Code Error Reporting

No production code error reporting is supported.

4 Integration

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

4.1 Scope of Delivery

The delivery of the VStdLib contains following static files. There are no dynamic files.

File Name	Description
vstdlib.c	This is the source file of the VStdLib.
vstdlib.h	This is the header file that contains the public API.
VStdLib_Cfg.h	This is the configuration header file, see chapter 6 for details.

Table 4-1 Static files

4.2 Include Structure

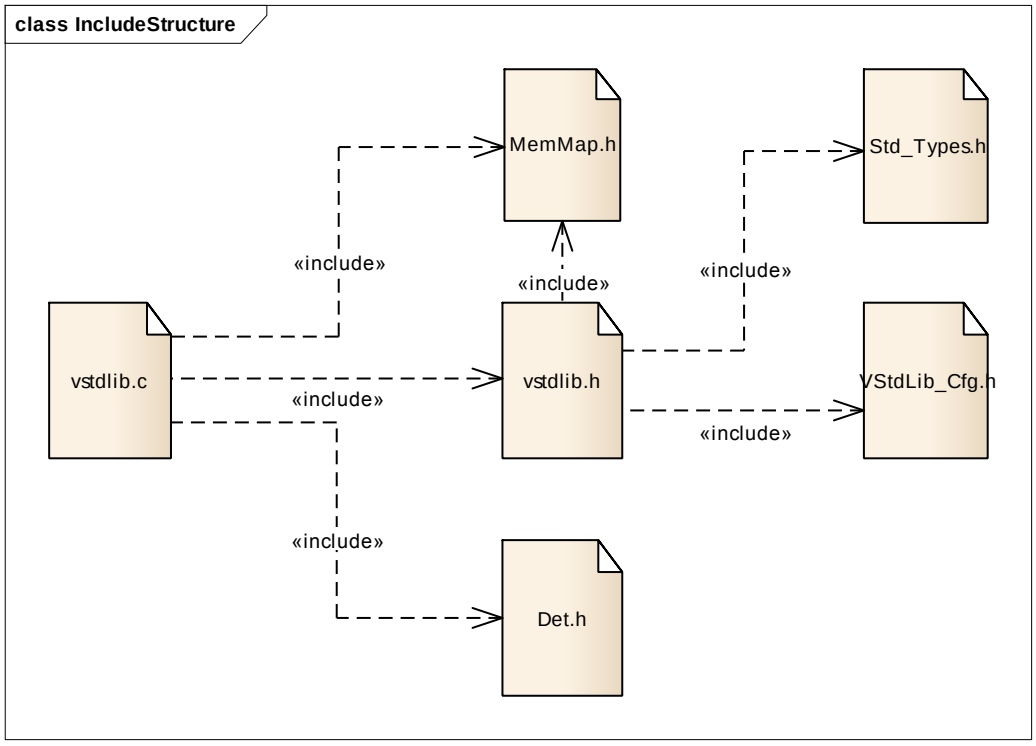

Figure 4-1 Include Structure

4.3 Critical Sections

No critical sections are implemented.

4.4 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions of the VStdLib and illustrates their assignment among each other.

Memory Mapping Sections \ Compiler Abstraction Definitions	VSTDLIB_CODE	VSTDLIB_APPL_VAR	VSTDLIB_VAR_FAR
VSTDLIB_START_SEC_CODE VSTDLIB_STOP_SEC_CODE	■		

Table 4-2 Compiler Abstraction and Memory Mapping

The VStdLib declares no global variables or constants thus only a general CODE section is provided for the memory mapping that is abstracted by `VSTDLIB_CODE`.

Due to the AUTOSAR compiler and memory abstraction the location of individual data and the width of pointers are not known during development time. Hence the compiler abstraction definition `VSTDLIB_APPL_VAR` refers to variables which are defined by the application and not handled by the VStdLib memory mapping.

The function implementation of all memory manipulation services as described in section 5.2 abstracts all pointer arguments (independently of the target type) with `VSTDLIB_VAR_FAR` to allow that all data can be handled by far accesses. The macro implementation performs an in-place access, thus the unaltered pointer class of the caller is used.

Note

For most 32bit MCUs it is not necessary to give any qualifiers with `VSTDLIB_VAR_FAR` to change the pointer class as these platforms commonly use 32bit pointers that can address the whole memory by default. Please note that both RAM and ROM are accessed with this qualifier and an external implementation (see section 6.2.1) has to be used if distinct pointers have to be qualified individually on the used target platform.

4.5 Integration Hints

- Avoid overlapping destination and source memory areas when using the copy services as this may lead to unexpected results.
- Consider side effects if macros are used: If either destination or source pointer is retrieved using a call to a function, this function might be invoked for nCnt times. It is recommended to use temporary pointers to avoid any side effects.
- The VStdLib is optimized for 32-bit MCUs. When using a controller with a smaller bit width it is highly recommended to use an external implementation that is optimized for this architecture (see `VSTDLIB_USE_LIBRARY_FUNCTIONS` in section 6.2.1). If large data support is not necessary define each `VSTDLIB_RUNTIME_OPTIMIZATION` and `VSTDLIB_SUPPORT_LARGE_DATA` to `STD_OFF` as an alternative.

5 API Description

See Figure 2-2 for an interfaces overview.

5.1 Type Definitions

The nCnt parameter of all memory manipulation services is of type `VStdLib_CntType` that equals uint32_least if `VSTDLIB_SUPPORT_LARGE_DATA` is defined to `STD_ON` (default setting) or uint16_least if it is explicitly defined to `STD_OFF`.

5.2 Services provided by VStdLib

5.2.1 VStdLib_GetVersionInfo

Prototype	
`void` **`VStdLib_GetVersionInfo`** `(Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [out]	Pointer to where to store the version information, must not be NULL.
Return code	
void	none
Functional Description	
Returns the version information of this module. Returns version information, vendor ID and AUTOSAR module ID of the component.	
Particularities and Limitations	
The parameter 'versioninfo' has to be valid and reference an object of type Std_VersionInfoType. Configuration Variant(s): `VSTDLIB_VERSION_INFO_API == STD_ON`	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-1 VStdLib_GetVersionInfo

5.2.2 VStdLib_MemClr

Prototype	
`void` **`VStdLib_MemClr`** `(void *pDst, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Pointer to the memory location to be initialized, must not be NULL.
nCnt [in]	Number of bytes to initialize, pDst must be valid for this amount.
Return code	
void	none
Functional Description	
Initializes memory to zero. Sets nCnt bytes starting at pDst to zero.	
Particularities and Limitations	
The parameters 'pDst' and 'nCnt' have to define a valid memory area. Configuration Variant(s): This service is implemented externally if `VSTDLIB_USE_LIBRARY_FUNCTIONS == STD_ON`, else it is realized by a call to VStdLib_MemSet() with 'nPattern' == 0. The compatible definition VStdLib_MemClrLarge() exists additionally (and solely) if `VSTDLIB_SUPPORT_LARGE_DATA == STD_ON`.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-2 VStdLib_MemClr

5.2.3 VStdLib_MemClrMacro

Prototype	
`void VStdLib_MemClrMacro (AnyPtrType *pDst, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Any typed pointer to the memory location to be initialized, must be aligned corresponding to its type and not be NULL.
nCnt [in]	Number of blocks to initialize, pDst must be valid for this amount.
Return code	
void	none
Functional Description	
Initializes memory to zero (macro implementation). Sets nCnt blocks starting at pDst to zero (block-size is given by the type of pDst).	
Particularities and Limitations	
The parameters 'pDst' and 'nCnt' have to define a valid memory area.	
Call context	
> ANY > This macro is Synchronous > This macro is Reentrant	

Table 5-3 VStdLib_MemClrMacro

5.2.4 VStdLib_MemSet

Prototype	
`void VStdLib_MemSet (void *pDst, uint8 nPattern, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Pointer to the memory location to be initialized, must not be NULL.
nPattern [in]	The character to be used to initialize the memory.
nCnt [in]	Number of bytes to initialize, pDst must be valid for this amount.
Return code	
void	none
Functional Description	
Initializes memory to a specified pattern. Sets nCnt bytes starting at pDst to the character nPattern.	
Particularities and Limitations	
The parameters 'pDst' and 'nCnt' have to define a valid memory area. Configuration Variant(s): This service is implemented externally if `VSTDLIB_USE_LIBRARY_FUNCTIONS == STD_ON`. The compatible definition VStdLib_MemSetLarge() exists additionally (and solely) if `VSTDLIB_SUPPORT_LARGE_DATA == STD_ON`.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-4 VStdLib_MemSet

5.2.5 VStdLib_MemSetMacro

Prototype	
`void VStdLib_MemSetMacro (AnyPtrType *pDst, AnyIntType nPattern, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Any typed pointer to the memory location to be initialized, must be aligned corresponding to its type and not be NULL.
nPattern [in]	The pattern to be used to initialize the memory (consider the correlation between its type and the type of pDst).
nCnt [in]	Number of blocks to initialize, pDst must be valid for this amount.
Return code	
void	none
Functional Description	
Initializes memory to a specified pattern (macro implementation). Sets nCnt blocks starting at pDst to nPattern (block-size is given by the type of pDst).	
Particularities and Limitations	
The parameters 'pDst' and 'nCnt' have to define a valid memory area.	
Call context	
> ANY > This macro is Synchronous > This macro is Reentrant	

Table 5-5 VStdLib_MemSetMacro

5.2.6 VStdLib_MemCpy

Prototype	
`void` **`VStdLib_MemCpy`** `(void *pDst, const void *pSrc, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Pointer to the memory location to copy to, must not be NULL.
pSrc [in]	Pointer to the memory location to copy from, must not be NULL.
nCnt [in]	Number of bytes to copy, pDst must be valid for this amount.
Return code	
void	none
Functional Description	
Copies data from one memory location to another. Copies nCnt bytes starting at pSrc to another memory location starting at pDst.	
Particularities and Limitations	
The parameters 'pDst' and 'nCnt' have to define a valid memory area. Configuration Variant(s): This service is implemented externally if `VSTDLIB_USE_LIBRARY_FUNCTIONS == STD_ON`. The compatible definition VStdLib_MemCpyLarge() exists additionally (and solely) if `VSTDLIB_SUPPORT_LARGE_DATA == STD_ON`.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-6 VStdLib_MemCpy

5.2.7 VStdLib_MemCpy16

Prototype	
`void` **`VStdLib_MemCpy16`** `(uint16 *pDst, const uint16 *pSrc, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Pointer to the memory location to copy to, 16-bit aligned and not NULL.
pSrc [in]	Pointer to the memory location to copy from, 16-bit aligned and not NULL.
nCnt [in]	Number of 16-bit blocks to copy, pDst must be valid for this amount.
Return code	
void	none
Functional Description	
Copies data from one memory location to another. Copies nCnt 16-bit blocks starting at pSrc to another memory location starting at pDst.	
Particularities and Limitations	
The parameters 'pDst' and 'nCnt' have to define a valid memory area. Configuration Variant(s): This service is implemented externally if `VSTDLIB_USE_LIBRARY_FUNCTIONS == STD_ON`. The compatible definition VStdLib_MemCpy16Large() exists additionally (and solely) if `VSTDLIB_SUPPORT_LARGE_DATA == STD_ON`.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-7 VStdLib_MemCpy16

5.2.8 VStdLib_MemCpy32

Prototype	
`void` **`VStdLib_MemCpy32`** `(uint32 *pDst, const uint32 *pSrc, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Pointer to the memory location to copy to, 32-bit aligned and not NULL.
pSrc [in]	Pointer to the memory location to copy from, 32-bit aligned and not NULL.
nCnt [in]	Number of 32-bit blocks to copy, pDst must be valid for this amount.
Return code	
void	none
Functional Description	
Copies data from one memory location to another. Copies nCnt 32-bit blocks starting at pSrc to another memory location starting at pDst.	
Particularities and Limitations	
The parameters 'pDst' and 'nCnt' have to define a valid memory area. Configuration Variant(s): This service is implemented externally if `VSTDLIB_USE_LIBRARY_FUNCTIONS == STD_ON`. The compatible definition VStdLib_MemCpy32Large() exists additionally (and solely) if `VSTDLIB_SUPPORT_LARGE_DATA == STD_ON`.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-8 VStdLib_MemCpy32

5.2.9 VStdLib_MemCpy_s

Prototype	
`void VStdLib_MemCpy_s (void *pDst, VStdLib_CntType nDstSize, const void *pSrc, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Pointer to the memory location to copy to, must not be NULL.
nDstSize [in]	Maximum number of bytes to modify in the destination (typically the size of the destination object).
pSrc [in]	Pointer to the memory location to copy from, must not be NULL.
nCnt [in]	Number of bytes to copy.
Return code	
void	none
Functional Description	
Verifies the destination size and copies data from one memory location to another. Uses VStdLib_MemCpy() to copy nCnt bytes starting at pSrc to another memory location starting at pDst, if nDstSize is greater than or equal to nCnt.	
Particularities and Limitations	
The parameters 'pDst' and 'nDstSize' have to define a valid memory area. Configuration Variant(s): The compatible definition VStdLib_MemCpyLarge_s() exists additionally (and solely) if `VSTDLIB_SUPPORT_LARGE_DATA == STD_ON`.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-9 VStdLib_MemCpy_s

5.2.10 VStdLib_MemCpyMacro

Prototype	
`void VStdLib_MemCpyMacro (AnyPtrType *pDst, AnyPtrType *pSrc, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Any typed pointer to the memory location to copy to, must be aligned corresponding to its type and not be NULL.
pSrc [in]	Any typed pointer to the memory location to copy from, must be aligned corresponding to its type and not be NULL.
nCnt [in]	Number of blocks to copy, pDst must be valid for this amount.
Return code	
void	none
Functional Description	
Copies data from one memory location to another (macro implementation). Copies nCnt blocks starting at pSrc to another memory location starting at pDst (block-size is given by the type of pDst).	
Particularities and Limitations	
The parameters 'pDst' and 'nCnt' have to define a valid memory area.	
Call context	
> ANY > This macro is Synchronous > This macro is Reentrant	

Table 5-10 VStdLib_MemCpyMacro

5.2.11 VStdLib_MemCpyMacro_s

Prototype	
`void` **`VStdLib_MemCpyMacro_s`** `(AnyPtrType *pDst, VStdLib_CntType nDstSize, AnyPtrType *pSrc, VStdLib_CntType nCnt)`	
Parameter	
pDst [out]	Any typed pointer to the memory location to copy to, must be aligned corresponding to its type and not be NULL.
nDstSize [in]	Maximum number of blocks to modify in the destination (typically the size of the destination object).
pSrc [in]	Any typed pointer to the memory location to copy from, must be aligned corresponding to its type and not be NULL.
nCnt [in]	Number of blocks to copy.
Return code	
void	none
Functional Description	
Verifies the destination size and copies data from one memory location to another (macro implementation). Uses VStdLib_MemCpyMacro() to copy nCnt blocks starting at pSrc to another memory location starting at pDst (block-size is given by the type of pDst), if nDstSize is greater than or equal to nCnt.	
Particularities and Limitations	
The parameters 'pDst' and 'nDstSize' have to define a valid memory area.	
Call context	
> ANY > This macro is Synchronous > This macro is Reentrant	

Table 5-11 VStdLib_MemCpyMacro_s

5.3 Services used by VStdLib

The following table lists used services that are provided by other components. For details about prototypes and functionality refer to the documentation of the providing component.

Component	API
DET	`Det_ReportError()`
(Compiler) Library	Refer to section 6.2.2

Table 5-12 Services used by VStdLib

6 Configuration

The Vector Standard Library is solely configured manually in a header file.

6.1 Configuration Variants

The VStdLib supports following configuration variants:

> VARIANT-PRE-COMPILE

6.2 Manual Configuration in Header File

The configuration of the VStdLib is done statically within the file “VStdLib_Cfg.h”.

6.2.1 General configuration

Attribute Name	Values Default value is typed bold	Description
VSTDLIB_USE_LIBRARY_FUNCTIONS	STD_ON **STD_OFF**	If set to STD_ON all memory manipulation routines are mapped to external functions (e.g. compiler libraries or other implementations that are optimized for the target platform). This requires additional configuration, refer to section 6.2.2. If set to STD_OFF generic functions provided by the VStdLib are used.
VSTDLIB_RUNTIME_OPTIMIZATION	**STD_ON** STD_OFF	If set to STD_ON optimized routines are used to increase the performance of the memory manipulation functions (increases code size). If set to STD_OFF code efficient routines are used that increase runtime. This setting is only relevant if VSTDLIB_USE_LIBRARY_FUNCTIONS == STD_OFF.
VSTDLIB_USE_JUMPTABLES	**STD_ON** STD_OFF	If set to STD_ON jump tables are used to increase the performance of the memory manipulation functions (runtime efficient in general). If set to STD_OFF the jump tables are replaced by loops. Use this setting if the compiler generates no efficient code for the jump tables. This setting is only relevant if VSTDLIB_USE_LIBRARY_FUNCTIONS == STD_OFF and VSTDLIB_RUNTIME_OPTIMIZATION == STD_ON.
VSTDLIB_DEV_ERROR_DETECT	STD_ON **STD_OFF**	If set to STD_ON the development error detection is enabled. In this case the pointer arguments of all global module functions provided by the VStdLib are checked. If any NULL_PTR is passed, these functions return without performing any action.

VSTDLIB_DEV_ERROR_REPORT	STD_ON **STD_OFF**	If set to STD_ON the development error reporting is enabled (requires VSTDLIB_DEV_ERROR_DETECT == STD_ON). In this case the function Det_ReportError() is called if any error is detected.
VSTDLIB_VERSION_INFO_API	STD_ON **STD_OFF**	If set to STD_ON the function VStdLib_GetVersionInfo() is provided.
VSTDLIB_DUMMY_STATEMENT(v)	e.g. (v) = (v) **(void)(v)**	Expression that is used for dummy statements to avoid compiler warnings about unused identifiers. Leave this definition empty to disable the usage of dummy statements.

Table 6-1 General configuration

6.2.2 Additional configuration when using library functions

If VSTDLIB_USE_LIBRARY_FUNCTIONS == STD_ON it is necessary to specify library functions to be used for the memory manipulations. See the corresponding section in “VStdLib_Cfg.h” for details and an example mapping.

Caution

If the external functionality is not able to handle more than 65535 bytes it is necessary to define VSTDLIB_SUPPORT_LARGE_DATA to STD_OFF.

It has to be ensured that the specified functions are able to copy from and to all memory locations independently of the pointer length. The specified functions must behave synchronously.

7 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MCU	Microcontroller Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
SWS	Software Specification
VStdLib	Vector Standard Library

Table 7-1 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com