VStdLib

Technical Reference

Vector Standard Library

Version 1.6

Authors	Patrick Markl
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Patrick Markl	2008-02-06	1.0	Creation, merge from Application Note
Patrick Markl	2008-10-31	1.3	Fixed document version
Patrick Markl	2008-11-07	1.4	Added information about mixed VStdLib versions
Patrick Markl	2009-07-02	1.5	Updated assertion codes Added chapter about QNX OS
Patrick Markl	2009-10-16	1.6	Described inclusion of OSEK OS header file

Table 1-1 History of the Document

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

Contents

Illustrations

Tables

2 Introduction

The basic idea of the VStdLib is to provide standard functionality to different Vector components. This includes memory copy and interrupt locking functions. The VStdLib is also a means to provide special memory copy functions, which are not available by compiler libraries to the communication components.

The VStdLib is designed to be used by Vector communication components only. So the customer integrating the communication stack will probably not notice the VStdLib at all.

The VStdLib is highly hardware specific. This means some functions are only available on certain platforms. This includes special copy functions for far, near or huge memory. A hardware specific VStdLib does not necessarily implement all possible copy function for the corresponding hardware platform, but only the functions required by the Vector software components.

The VStdLib also provides interrupt lock functions. This feature depends on the CANbedded CAN driver's Reference Implementation (RI). The RI defines a certain feature set to be supported by the CAN driver. CAN drivers which implement RI1.5 and higher do not provide an interrupt locking mechanism as previously implementations did (CanGlobalInterruptDisable/CanGlobalInterruptRestore). These driver require a VStdLib to provide this functionality, although the API CanGlobalInterruptDisable and CanGlobalInterruptRestore remains for compatibility. Please refer to the CANbedded CAN driver technical reference for more information about your specific CAN driver.

3 Functional Description

The VStdLib provides basically two main functionalities to Vector communication stack components.

- Functions to copy memory
- Functions to lock/unlock interrupts

Interrupt locking functionality is required for CAN drivers with a reference inplementation 1.5 or higher or MICROSAR stacks. Functions to copy memory are always provided. The following table describes the API of the VStdLib which is used internally.

API Function	Description
VStdMemSet	Initializes default RAM memory to a certain character.
VStdMemNearSet	Initializes near RAM memory to a certain character.
VStdMemFarSet	Initializes far RAM memory to a certain character.
VStdMemClr	Clears default RAM to zero.
VStdMemNearClr	Clears near RAM to zero.
VStdMemFarClr	Clears far RAM to zero.
VStdMemCpyRamToRam	Copies default RAM to default RAM.
VStdMemCpyRomToRam	Copies default ROM to default RAM.
VStdMemCpyRamToNearRam	Copies default RAM to near RAM.
VStdMemCpyRomToNearRam	Copies default ROM to near RAM.
VStdMemCpyRamToFarRam	Copies default RAM to far RAM.
VStdMemCpyRomToFarRam	Copies default ROM to far RAM.
VStdMemCpy16RamToRam	Copies default RAM to default RAM. The copying is performed 16 bit wise.
VStdMemCpy16RamToFarRam	Copies default RAM to far RAM. The copying is performed 16 bit wise.
VStdMemCpy16FarRamToRam	Copies far RAM to default RAM. The copying is performed 16 bit wise.
VStdMemCpy32RamToRam	Copies default RAM to default RAM. The copying is performed 32 bit wise.
VStdMemCpy32RamToFarRam	Copies default RAM to far RAM. The copying is performed 32 bit wise.
VStdMemCpy32FarRamToRam	Copies far RAM to default RAM. The copying is performed 32 bit wise.

Info
The API functions described in the following table contain “near” and “far” as parts of the API names. The meaning of near and far depends on the platform and the functions are not necessarily implemented to work on near or far memory.

The wording “default RAM” or “default ROM” means RAM or ROM data which is not explicitely declared to be near or far. These data inherit the current compiler memory model settings.

Caution
One cannot be sure that a far to near copy routine (for instance) implements really a copying from far to near. This depends on the platform and the supported memory models of the communication stack!

4 Integration

The integration of the VStdLib is straightforward. As the other Vector components the VStdLib is to be configured by means of the configuration tool. The configuration is described in the next chapter. Chapter 6 describes some callback function which have to be implemented by the application depending on the configuration.

In order to integrate the VStdLib simply add the file vstdlib.c as the other Vector software components to the compile and link list.

4.1 CANbedded Particularities

If the VStdLib is integrated as part of a CANbedded stack the initialization of the VStdLib is performed by the CAN or LIN driver automatically.

4.2 MICROSAR Particularities

If a MICROSAR stack is integrated the VStdLib is not initialized by any of the software components. The application has to take care that the VStdLib's initialization function is called before any other MICROSAR API function is called. The following code example shows how to initialize the VStdLib.

```
void InitRoutine(void)
{
  ...

  /* Initialize the VstLib */
  VStdInitPowerOn();

  /* Perform initialization of the other software components */
  ...

}
```

4.3 Mixing VStdLib Versions

If you receive different software component packages from Vector, it is possible that these packages contain different VStdLib versions. A general rule is to use the VStdLib which was received with the CAN driver package. In case you encounter any incompatibilities or have questions, please contact the Vector support.

5 Configuration

It depends on the used communication software if the VStdLib has configurable options. The configuration tool provides options for the VStdLib if (as previously described) the used CAN driver has RI1.5 or higher or a MICROSAR stack is used.

The VStdLib is configured by means of the configuration tool. CANGen provides an own dialog named VStdLib to configure this component. In GENy these configuration options are found in the hardware options dialog. The following two pictures show the configuration dialogs in both tools. The first picture shows the dialog in GENy.

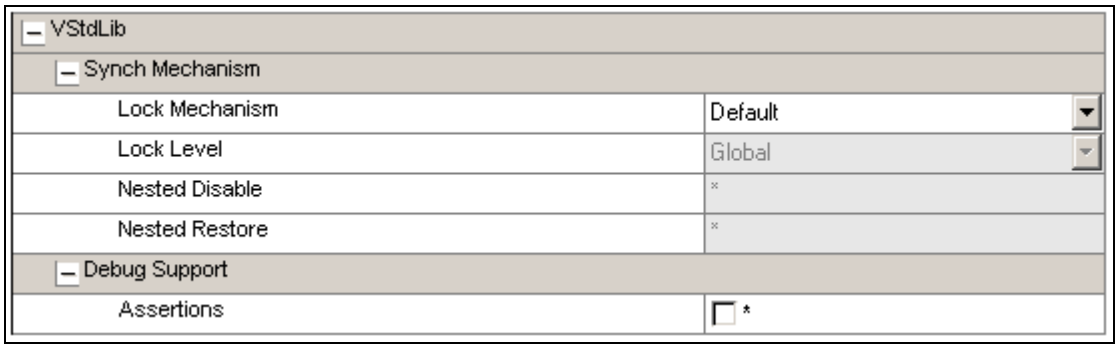

Figure 5-1 VStdLib configuration dialog in GENy

The next picture shows the configuration dialog of the VStdLib in the CANGen tool.

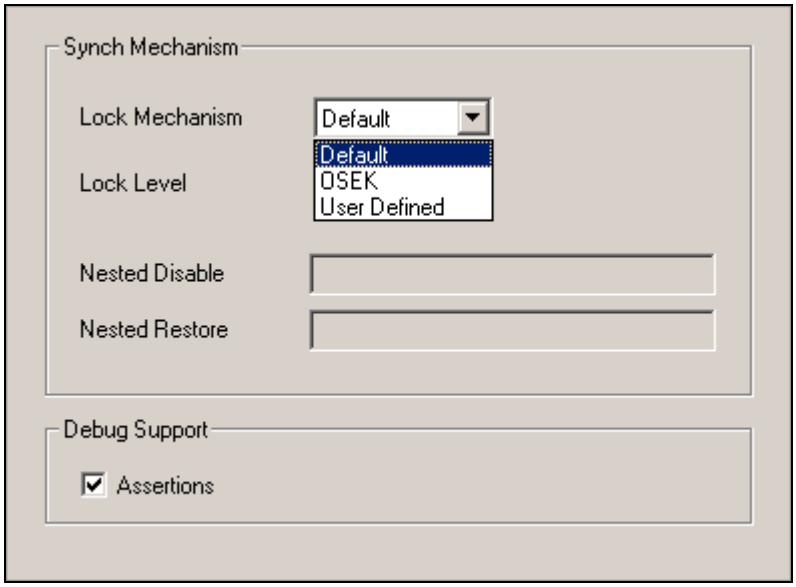

Figure 5-2 VStdLib configuration dialog in CANGen

The following table describes the different configuration items of the VStdLib dialogs.

Configuration Item	Description
Lock Mechanism	One can select the interrupt lock mechanism used by the VStdLib. The VStdLib provides four different types: Default, LockLevel, OSEK and User. The setting “OSEK” covers OSEK OS and AUTOSAR OS.
Lock Level	If LockLevel was selected as Lock Mechanism one can select the interrupt lock level used for CANbedded components. The VStdLib will lock interrupts up to the specified level. Higher level interrupts remain unlocked.
Nested Disable	If “User” was selected as Lock Mechanism the user can type in the name of the user defined function which lock the interrupts.
Nested Restore	If “User” was selected as Lock Mechanism the user can type in the name of the user defined function which restores the interrupts to their previous state.
Assertions	If this checkbox is enabled the VStdLib will call an assertion function in case of a fatal error.

If one configures the option Lock Mechanism as “User” two function have to be provided. The prototype of both functions is:

```
void UserFunction(void)
```

The first function entered under the option “Nested Disable” is expected to disable interrupts and store the current context of the interrupt state (Current I-Flag, Current Interrupt Lock Level, etc.) to a global variable.

The second function entered under “Nested Restore” is expected to restore the interrupts to the previous state using the value which was saved in the context of “Nested Disable”.

Caution
The two functions must not implement a disable/enable mechanism but a disable/restore mechanism! Otherwise it may happen that interrupts remain locked until the next Power On reset.

The configuration option “User” can be used instead of the previously known CAN driver option “Interrupt Control by Application”.

5.1 Usage with OSEK OS

If OSEK OS is configured as lock mechanism the VStdLib needs to include the header of the OSEK OS. Since the header of the OSEK OS may have different names depending on the OS supplier the VStdLib includes always the same header file. This header file has to be provided by the user. The file’s name must be os.h. If the OSEK OS already provides a file named like this nothing needs to be done (as long it provides the required prototypes). Otherwise the user has to create a header file named os.h and include the corresponding OSEK OS file.

If the VStdLib is configued to lock/unlock interrupts by means of OSEK OS the following OSEK functions are called to implement the lock:

- SuspendAllInterrupts
- ResumeAllInterrupts

5.2 Usage with QNX OS

In case the VStdLib is used in a QNX OS environment two options for interrupt locking are available. Depending on the way CAN interrupts are handled (ISR, Interrupt Thread) different locking mechanisms are needed.

In case the CAN interrupts are handled by an interrupt thread the locking is done by means of mutexes. These are not implemented by the VStdLib but the QWrap component. Please refer to Figure 5-3 which shows the locking configuration in this case.

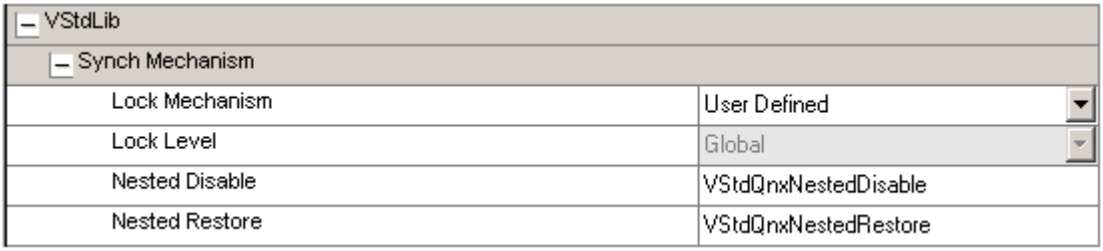

Figure 5-3 Locking configuration in case CAN events are handled in an interrupt thread

User defined locking is also required if the VStdLib is configured to run in the context of the QNX startup code (minidriver). The MdWrap component provides the same callbacks as shown in Figure 5-3 walso in that case. QWrap and MdWrap implement the same callback names in case the same tool configuration is used to generate for the minidriver and the fulldriver.

In case CAN interrupts are handled by a CAN interrupt routine the VStdLib locking functions disable/restore the global interrupt flag. Mutexes are not used. In this case the configuration of the VStdLib must be set to Default as shown in Figure 5-4.

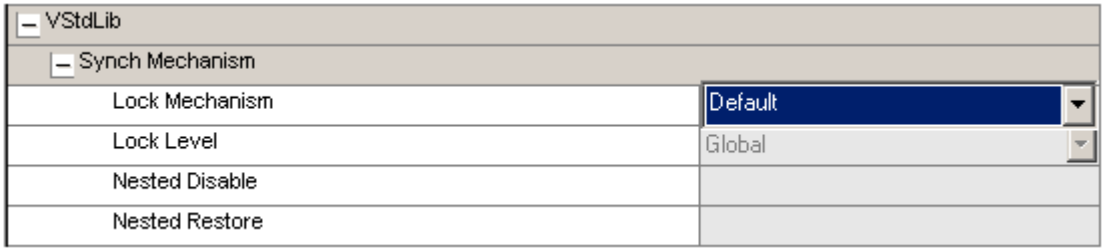

Figure 5-4 Configuration of the VStdLib in case of handling CAN events on interrupt level

6 API Description

This chapter describes the API of the VStdLib.

6.1 Initialization

VStdInitPowerOn

Prototype	
`void VStdInitPowerOn(void)`	
Parameter	
None	-
Return code	
None	-
Functional Description	
Initializes the VStdLib component.	
Particularities and Limitations	
■ This function must be called once after PowerOn reset. This function must not be re-called later. ■ The application must not call any other Vector communication stack function before this function. This includes also the other VStdLib functions. In other words: The function VStdInitPowerOn must be the very first Vector function being called after startup. Depending on the architecture the caller must take care about interruptions of the init function which may violate this requirement (OS task which calls another Vector function). ■ If the CANbedded stack is used this function is automatically called by the CAN or LIN driver. ■ In a MICROSAR communication stack is integrated this function has to be called by the application before any other MICROSAR API function is called.	
Call context	
■ Task context	

6.2 Interrupt Control

VStdSuspendAllInterrupts

Prototype	
`void VStdSuspendAllInterrupts(void)`	
Parameter	
None	-
Return code	
None	-
Functional Description	
Disables all interrupts or locks interrupts to a certain lock level. This depends on the hardware platform. The way interrupts are locked is to be configured in the configuration tool.	
Particularities and Limitations	
■ This function is designed to be called in a nested way (except User Interrupt Control is configured). ■ Every call to VStdSuspendAllInterrupts must have a corresponding call to VStdResumeAllInterrupts. ■ If User Interrupt Control is configured this function is redirected to the functions specified by the user in the configuration tool. No nesting counter is implemented. The user may have to take special care to handle this behavior. ■ Depending on the used OS this function may use either a global lock or a mutex.	
Call context	
■ No limitation	

VStdResumeAllInterrupts

Prototype	
`void VStdResumeAllInterrupts(void)`	
Parameter	
None	-
Return code	
None	-
Functional Description	
Resumest the interrupts which were locked using the API function VStdSuspendAllInterrupts.	
Particularities and Limitations	
■ This function must not be called without a corresponding call to VStdSuspendAllInterrupts. ■ If User Interrupt Control is configured this function is redirected to the functions specified by the user in the configuration tool. No nesting counter is implemented. The user may have to take special care to handle this behavior. ■ Depending on the used OS this function may use either a global lock or a mutex.	
Call context	

■ No limitation

6.3 Memory Functions

This chapter describes not all functions, as they do all the same except for the memory qualifiers of the source and destionation locations.

VStdMemSet

Prototype	
`void VStdMemSet (void *pDest, vuint8 nPattern, vuint16 nCnt)`	
Parameter	
pDest	The start address in memory which is to be initialized using the given character.
nPattern	The character to be used to fill nCnt Bytes starting at address pDest.
nCnt	The number of Bytes to be filled using the given character.
Return code	
None	-
Functional Description	
This function fills nCnt Bytes starting at address pDest in memory using the character nPattern.	
Particularities and Limitations	
■ This function exists in three variants (default, near and far memory). Depending on the plattform these function may be implemented differently	
Call context	
■ No restriction	

VStdMemSet

Prototype	
`void VStdMemClr (void *pDest, vuint16 nCnt)`	
Parameter	
pDest	The start address in memory which is to be initialized to zero.
nCnt	The number of Bytes to be set to zero.
Return code	
None	-
Functional Description	
This function initializes nCnt Bytes starting at address pDest in memory to zero.	
Particularities and Limitations	
■ This function exists in three variants (default, near and far memory). Depending on the plattform these function may be implemented differently	

Call context
■ No restriction

VStdMemSet

Prototype	
`void VStdMemCpy (void *pDest, void *pSrc, vuint16 nCnt)`	
Parameter	
pDest	The destination address to which the data is copied.
pSrc	The source address from where the data is copied.
nCnt	The number of Bytes to be copied from start address pSrc to destination address pDest.
Return code	
None	-
Functional Description	
This function copies a block of nCnt Bytes starting at memory location pSrc to another memory location starting at address pDest.	
Particularities and Limitations	
■ This function exists in many variants. These variants implement different memory source and destination locations. ■ The implementation of these functions is highly plattform dependant ■ The memory blocks starting at pSrc and pDest must no overlap.	
Call context	
■ No restriction	

6.4 Callback Functions

ApplVStdFatalError

Prototype	
`void ApplVStdFatalError (vuint8 nErrorNumber)`	
Parameter	
nErrorNumber	This parameter describes the reason which caused the assertion.
Return code	
None	-
Functional Description	
This function is called by the VStdLib in case an assertion fails.	

Particularities and Limitations
■ This function is to be implemented by the application. ■ If this function returns to the caller, it is not ensured that the calling functions finished successfully. It has to be considered as a severe error if this function is called.
Call context
■ No restriction

7 Assertions

If the user enables the option “Assertions” the VStdLib component performs additional checks. If a checked condition fails the assertion function ApplVStdFatalError is called. The implementation of this function is to be provided by the application. This function has an errorcode parameter which describes the reason which caused the assertion. The following table describes these reasons.

Errorcode	Description
kVStdErrorIntDisableTooOften	The called VStdLib interrupt locking function exceeded a nesting call depth of 127.
kVStdErrorIntRestoreTooOften	The number of calls of the VStdLib interrupt unlocking function was higher than the number of nested interrupt lock calls.
kVStdErrorMemClrInvalidParameter	A memclr function of the VStdLib was called with an invalid pointer parameter
kVStdErrorMemCpyInvalidParameter	A memcpy function of the VStdLib was called with an invalid pointer parameter
kVStdErrorMemSetInvalidParameter	A memset function of the VStdLib was called with an invalid pointer parameter
kVStdErrorUnexpectedLockState	A interrupt restore function of the VStdLib is called but the interrupts are already unlocked. This state is not expected and must never occur.

Caution
Note that some of the assertions exist solely in case the VStdLib is configured to use no library functions. In that case the missing library functionality is implemented directly in the VStdLib source file.

Note
Not every assertion code is implemented on each hardware.

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
User configuration file	A text file which is appended by the configuration tool to the generated source files. If a user configuration file is required the technical reference describes it and the contents.

8.2 Abbreviations

Abbreviation	Description
RI	Reference implementation of the CANbedded CAN driver. The reference implementation defines the features a CAN driver must provide. From Reference Implementation 1.5 (RI1.5) the interrupt locking functionality of each CAN driver is moved to the VStdLib for the specific platform.
VStdLib	Vector Standard Library

9 Contact

Visit our website for more information on

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

www.vector-informatik.com