MICROSAR VX1000If

Technical Reference

VX1000 Interface

Version 1.2.0

Authors	Oliver Reineke, Hannes Haas
Status	Released

Document Information

History

Author	Date	Version	Remarks
Oliver Reineke	2015-08-10	1.0.0	Initial version
Hannes Haas	2015-09-10	1.0.1	Updated introduction
Oliver Reineke	2017-01-31	1.1.0	Added VX1000If_DynAddrSig_UpdateAddress
Oliver Reineke	2017-02-17	1.2.0	Added VX1000 Addon handling

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[2]	Vector Informatik	VX1000 Device Driver	
[3]	Vector Informatik	Application Note AN-IMC-1-016 VX1000: Getting Started with Nexus JTAG and MPC5554	V1.0.0
[4]	Vector Informatik	TechnicalReference_Supplement_VX1000If.pdf (obtainable via VXsupport@vector.com)	V1.0.0 or later

Scope of the Document

This technical reference describes the general use of the MICROSAR VX1000If.

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.

©2017, Vector Informatik GmbH

Contents

©2017, Vector Informatik GmbH

Illustrations

Tables

1 Component History

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

Component Version	New Features
1.00	Initial Version
1.01	Added VX1000If_DynAddrSig_UpdateAddress
1.02	Added reference to TechnicalReference_Supplement_VX1000If.pdf

Table 1-1 Component history

2 Introduction

This document describes the functionality and API of the AUTOSAR CDD module VX1000If.

Supported AUTOSAR Release*:	3, 4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	VX1000IF_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	VX1000IF_MODULE_ID	255 decimal (according to ref. [1])

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

2.1 Scope and Limitations

The main purpose of the VX1000If component is to disable the VX1000 driver by wrapping the platform- and derivate-specific VX1000 device driver services.

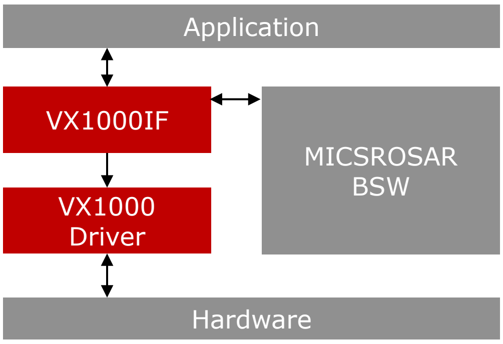


Figure 2-1 High-level Architecture of the VX1000 software stack (red)

Caution
The VX1000 driver APIs are not developed and tested for serial production. Therefore, if the VX1000 driver remains integrated in serial production conditions, in particular, within electronic control units of serial production vehicles (i) the VX1000 driver APIs must not be accessed by the application; and (ii) the API VX1000If_IsVX1000DriverAccessEnabled of the VX1000If component must return FALSE as described in chapter 5.2.1.

If the VX1000 driver is intended for development purposes, access by applications to the VX1000 driver may be enabled in a development environment by having the API VX1000If_IsVX1000DriverAccessEnabled return TRUE.

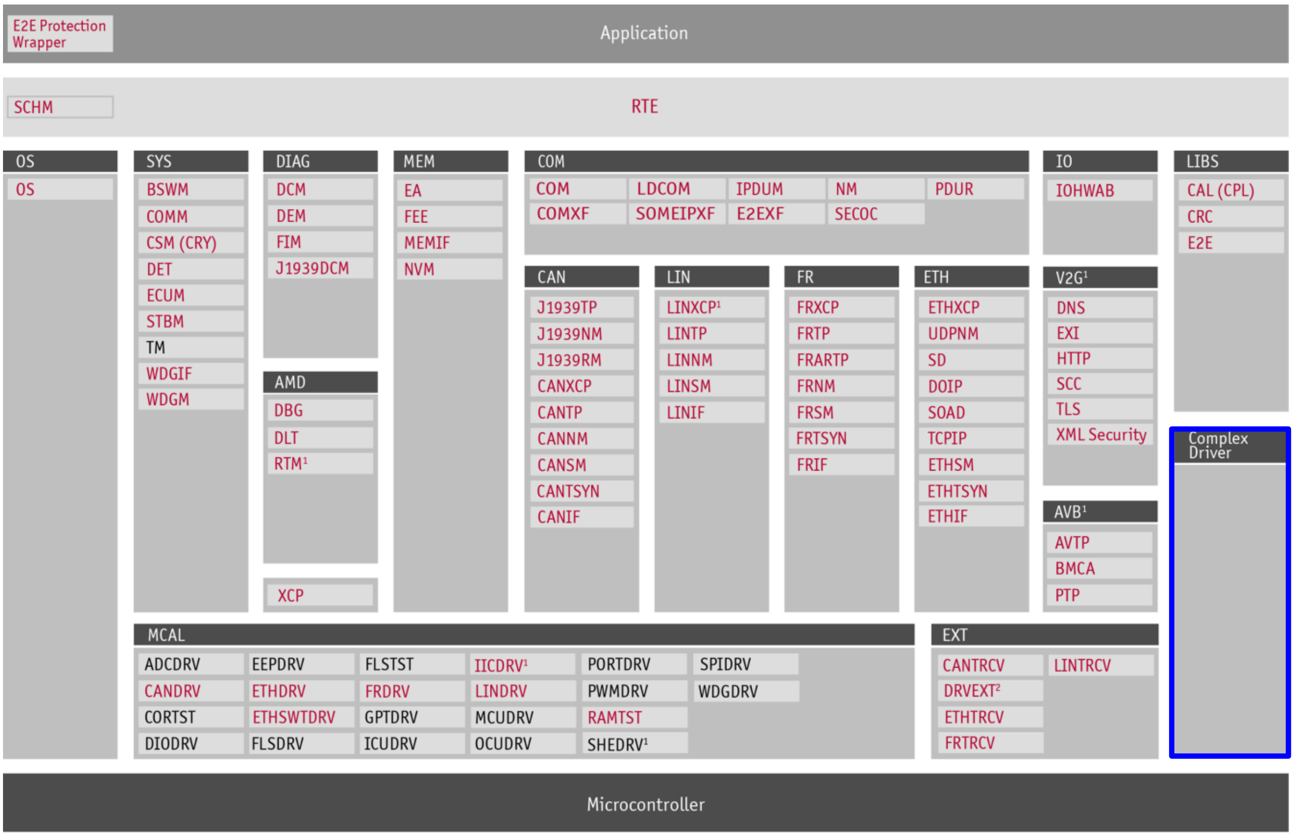
FAQ

The VX1000 system is a scalable solution with top performance for your measurement and calibration tasks. It can be used in the vehicle – both in the interior and in the engine compartment – on test benches and in the laboratory. The system forms the interface between the ECU and a measurement and calibration tool such as CANape. For high data throughput with minimal impact on ECU run-time, data is accessed over the microcontroller-specific data trace and debug ports.

The VX1000 base module is connected to the PC over XCP on Ethernet, an OEM-independent ASAM standard that is widely used in the automotive industry. The VX1000 measurement hardware is connected to the ECU via a POD (Plug-On device). Depending on the available microcontroller interface, either the data trace or a copying method can be used to acquire measurement data.

2.2 Architecture Overview

The following figure shows where the VX1000If is located in the AUTOSAR architecture.



[1] Available extensions for AUTOSAR
[2] Includes EXTADC, EEPEXT, FLSEXT, ETHSWTEXT and WDGEXT

Figure 2-2 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the VX1000If. These interfaces are described in chapter 5.

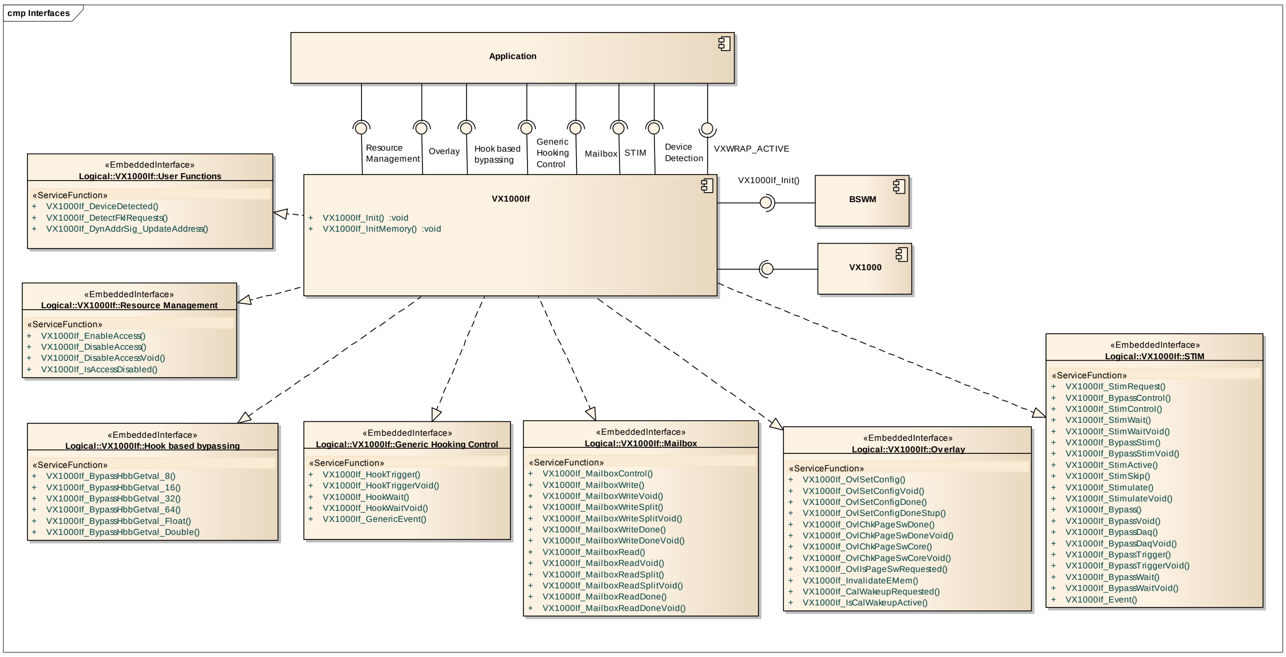

cmp Interfaces
Application
Resource Management
Overlay
Hook based bypassing
Generic Hooking Control
Mailbox
STIM
Device Detection
VXWRAP_ACTIVE
VX1000If_Init()
BSWM
VX1000
VX1000If
+ VX1000If_Init() :void
+ VX1000If_InitMemory() :void
«EmbeddedInterface»
Logical::VX1000If::User Functions
«ServiceFunction»
+ VX1000If_DeviceDetected()
+ VX1000If_DetectFklRequests()
+ VX1000If_DynAddrSig_UpdateAddress()
«EmbeddedInterface»
Logical::VX1000If::Resource Management
«ServiceFunction»
+ VX1000If_EnableAccess()
+ VX1000If_DisableAccess()
+ VX1000If_DisableAccessVoid()
+ VX1000If_IsAccessDisabled()
«EmbeddedInterface»
Logical::VX1000If::Hook based bypassing
«ServiceFunction»
+ VX1000If_BypassHbbGetval_8()
+ VX1000If_BypassHbbGetval_16()
+ VX1000If_BypassHbbGetval_32()
+ VX1000If_BypassHbbGetval_64()
+ VX1000If_BypassHbbGetval_Float()
+ VX1000If_BypassHbbGetval_Double()
«EmbeddedInterface»
Logical::VX1000If::Generic Hooking Control
«ServiceFunction»
+ VX1000If_HookTrigger()
+ VX1000If_HookTriggerVoid()
+ VX1000If_HookWait()
+ VX1000If_HookWaitVoid()
+ VX1000If_GenericEvent()
«EmbeddedInterface»
Logical::VX1000If::Mailbox
«ServiceFunction»
+ VX1000If_MailboxControl()
+ VX1000If_MailboxWrite()
+ VX1000If_MailboxWriteVoid()
+ VX1000If_MailboxWriteSplit()
+ VX1000If_MailboxWriteSplitVoid()
+ VX1000If_MailboxWriteDone()
+ VX1000If_MailboxWriteDoneVoid()
+ VX1000If_MailboxRead()
+ VX1000If_MailboxReadVoid()
+ VX1000If_MailboxReadSplit()
+ VX1000If_MailboxReadSplitVoid()
+ VX1000If_MailboxReadDone()
+ VX1000If_MailboxReadDoneVoid()
«EmbeddedInterface»
Logical::VX1000If::Overlay
«ServiceFunction»
+ VX1000If_OvlSetConfig()
+ VX1000If_OvlSetConfigVoid()
+ VX1000If_OvlSetConfigDone()
+ VX1000If_OvlSetConfigDoneStup()
+ VX1000If_OvlChkPageSwDone()
+ VX1000If_OvlChkPageSwDoneVoid()
+ VX1000If_OvlChkPageSwCore()
+ VX1000If_OvlChkPageSwCoreVoid()
+ VX1000If_OvlIsPageSwRequested()
+ VX1000If_InvalidateEMem()
+ VX1000If_CalWakeupRequested()
+ VX1000If_IsCalWakeupActive()
«EmbeddedInterface»
Logical::VX1000If::STIM
«ServiceFunction»
+ VX1000If_StimRequest()
+ VX1000If_BypassControl()
+ VX1000If_StimControl()
+ VX1000If_StimWait()
+ VX1000If_StimWaitVoid()
+ VX1000If_BypassStim()
+ VX1000If_BypassStimVoid()
+ VX1000If_StimActive()
+ VX1000If_StimSkip()
+ VX1000If_Stimulate()
+ VX1000If_StimulateVoid()
+ VX1000If_Bypass()
+ VX1000If_BypassVoid()
+ VX1000If_BypassDaq()
+ VX1000If_BypassDaqVoid()
+ VX1000If_BypassTrigger()
+ VX1000If_BypassTriggerVoid()
+ VX1000If_BypassWait()
+ VX1000If_BypassWaitVoid()
+ VX1000If_Event()

Figure 2-3 Interfaces to adjacent modules of the VX1000If

Applications access the services of the MICROSAR CDD modules directly.

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the VX1000If.

The following features are supported:

Supported Features
VX1000 Device Driver Abstraction

Table 3-1 Supported features

3.2 Initialization

After the (re)start of the ECU the VX1000If is in state “UNINIT”. In this state the VX1000If is not operable until the interface VX1000If_Init() is called.

VX1000If_Init() will change the state to “PRE_INIT”. Within this state:

- The start-up handshake with the VX1000 device driver can be triggered via VX1000If_InitAsyncStart() while the application hook VX1000If_IsVX1000DriverAccessEnabled returns TRUE.
- All other VX1000 API services will not be triggered by the VX1000If component.

VX1000If_InitAsyncStart() will change the state to “INITITIALIZED”.

Afterwards the VX1000If initialization sequence is finished and the VX1000 services are available via the VX1000If interface.

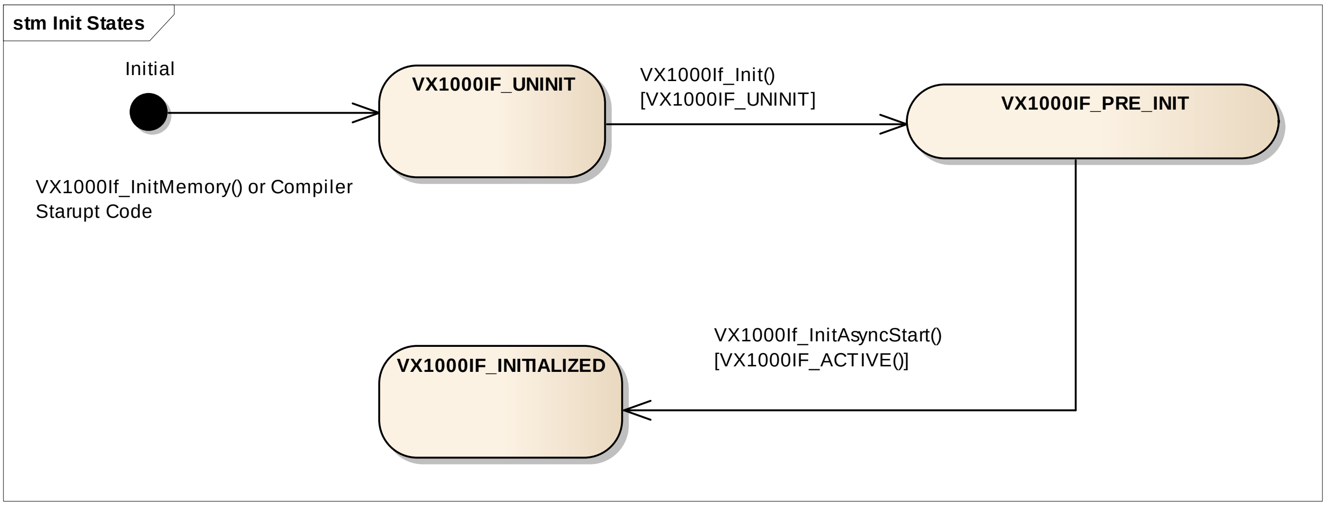

stm Init States
Initial
VX1000If_InitMemory() or Compiler Starupt Code
VX1000IF_UNINIT
VX1000If_Init() [VX1000IF_UNINIT]
VX1000IF_PRE_INIT
VX1000If_InitAsyncStart() [VX1000IF_ACTIVE()]
VX1000IF_INITIALIZED

Figure 3-1 VX1000If states

Caution
The application has to ensure that the timer for the VX1000 clock is up and running before calling VX1000If_InitAsyncStart() or subsequently VX1000If_InitAsyncEnd().

3.3 Error Handling

Neither the VX1000If component nor the VX1000 device driver support the AUTOSAR mechanisms for Development Error Reporting (DET) and Production Code Error Reporting (DEM).

The validity of the VX1000If service parameters must be ensured by the application.

In case the application calls a VX1000If non-void and non-hook based bypassing service while either

- the VX1000If state is not “INITIALIZED”
- or VX1000If_IsVX1000DriverAccessEnabled returned FALSE

the service returns VX1000IF_RET_E_NOT_OK and increments the global variable `VX1000If_ErrorCount` as error indication.

Error	Code	Description
255	VX1000IF_RET_E_NOT_OK	This error code is returned when non-void and non-hook based bypassing services are called while either: - the VX1000If state was not INITIALIZED - or VX1000If_IsVX1000DriverAccessEnabled returned FALSE

Table 3-2 Error return codes

©2017, Vector Informatik GmbH

4 Integration

This chapter gives necessary information for the integration of the MICROSAR VX1000If into an application environment of an ECU.

4.1 VX1000 Integration

For event-triggered, time-stamped data acquisition (DAQ), some additional program routines have to be added to the ECU software. There is only one function call required in the ECU software to trigger a particular DAQ event.

For data trace-based DAQ, triggering is done by a specific write access within the traced RAM. Up to 253 event channels can be used for data trace-based DAQ.

For copying-mechanism-based DAQ, the DAQ data is copied to an intermediate location driven by a DAQ transfer table. From here the data is read by the VX1000 using DMA capabilities of the hardware debugging modules. Up to 31 DAQ event channels can be used for copying-mechanism-based DAQ.

For a detailed description about the setup and code instrumentation for measurement data acquisition, for example via the Nexus JTAG interface of an MPC5554, refer to [3].

4.2 Scope of Delivery

The delivery of the VX1000If contains the files which are described in the chapters 4.2.1.

4.2.1 Static Files

File Name	Description
VX1000If.c	This is the source file of the VX1000 Interface.
VX1000If.h	This is the header file of the VX1000 Interface.

Table 4-1 Static files

5 API Description

For an interfaces overview please see Figure 2-3.

Note
The following API signature documentation in section 5.1 may partly differ from the actual implementation of the VX1000 device driver.

Refer to [2] for detailed API description of the VX1000 device driver.

5.1 Services provided by VX1000If

5.1.1 Startup and Shutdown

5.1.1.1 VX1000If_Init

Prototype	
void **VX1000If_Init** (void)	
Parameter	
void	-
Return code	
void	-
Functional Description	
Initializes all global VX1000 Interface data structures.	
Particularities and Limitations	
> None	

Table 5-1 VX1000If_Init

5.1.1.2 VX1000If_InitAsyncStart

Prototype	
void **VX1000If_InitAsyncStart** (void)	
Parameter	
void	-
Return code	
void	-
Functional Description	
Makes the VX1000 device driver detect an attached VX1000 device and perform a first handshake.	

<table>
<tr><th>Particularities and Limitations</th></tr>
<tr><td>&gt; VX1000If_Init() must be called</td></tr>
</table>

Table 5-2 VX1000If_InitAsyncStart

5.1.1.3 VX1000If_InitAsyncEnd

<table>
<tr><th colspan="2">Prototype</th></tr>
<tr><td colspan="2">void VX1000If_InitAsyncEnd (void)</td></tr>
<tr><th colspan="2">Parameter</th></tr>
<tr><td>void</td><td>-</td></tr>
<tr><th colspan="2">Return code</th></tr>
<tr><td>void</td><td>-</td></tr>
<tr><th colspan="2">Functional Description</th></tr>
<tr><td colspan="2">Makes the VX1000 device driver perform the second part of the handshake with an attached VX1000 device. Waits for end of handshake and can be used to synchronize the instrumentation on several cores.</td></tr>
<tr><th colspan="2">Particularities and Limitations</th></tr>
<tr><td colspan="2">&gt; VX1000If_InitAsyncStart must have been called</td></tr>
</table>

Table 5-3 VX1000If_InitAsyncEnd

5.1.1.4 VX1000If_PrepareSoftreset

<table>
<tr><th colspan="2">Prototype</th></tr>
<tr><td colspan="2">uint8 VX1000If_PrepareSoftreset (void)</td></tr>
<tr><th colspan="2">Parameter</th></tr>
<tr><td>void</td><td>-</td></tr>
<tr><th colspan="2">Return code</th></tr>
<tr><td>uint8</td><td>0 - reset procedure confirmed by tool
1 - handshake failed (measurement will not be resumed after the reset)
VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE</td></tr>
<tr><th colspan="2">Functional Description</th></tr>
<tr><td colspan="2">Makes the VX1000 device driver inform an attached VX1000 device about an upcoming software reset.</td></tr>
<tr><th colspan="2">Particularities and Limitations</th></tr>
<tr><td colspan="2">&gt; VX1000If_InitAsyncEnd must have been called</td></tr>
</table>

Table 5-4 VX1000If_PrepareSoftreset

5.1.1.5 VX1000If_PrepareSoftresetVoid

<table>
<tr><th>Prototype</th></tr>
<tr><td>void VX1000If_PrepareSoftresetVoid (void)</td></tr>
</table>

Parameter	
void	-
Return code	
void	-
Functional Description	
Makes the VX1000 device driver inform an attached VX1000 device about an upcoming software reset.	
Particularities and Limitations	
> VX1000If_InitAsyncEnd must have been called	

Table 5-5 VX1000If_PrepareSoftresetVoid

5.1.2 STIM

5.1.2.1 VX1000If_StimControl

Prototype	
void **VX1000If_STIM_CONTROL** (void)	
Parameter	
void	-
Return code	
void	-
Functional Description	
DEPRECATED: VX1000If_BypassControl should be used for new projects! Must be cyclically called by the application if STIM/Bypassing is used. Makes the VX1000 device driver perform bypassing management tasks, like globally starting and stopping bypassing operation.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-6 VX1000If_StimControl

5.1.2.2 VX1000If_BypassControl

Prototype	
void **VX1000If_BypassControl** (void)	
Parameter	
void	-
Return code	
void	-
Functional Description	
Must be cyclically called by the application if STIM/Bypassing is used. Makes the VX1000 driver perform bypassing management tasks, like globally starting and stopping bypassing operation.	

Particularities and Limitations
> VX1000If_InitAsyncStart must have been called.

Table 5-7 VX1000If_BypassControl

5.1.2.3 VX1000If_StimRequest

Prototype	
void **VX1000If_StimRequest**(uint8 stim_event)	
Parameter	
stim_event	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
void	-
Functional Description	
Makes the VX1000 device driver request a specific STIM data set associated to event channel stim_event.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-8 VX1000If_StimRequest

5.1.2.4 VX1000If_StimWait

Prototype	
uint8 **VX1000If_StimWait**(uint8 stim_event, uint32 timeout_us)	
Parameter	
stim_event	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
timeout_us	timeout in microseconds, starting from related call to VX1000If_StimRequest
Return code	
uint8	0 - data has arrived before timeout or timeout has occured but data has still been copied successfully 1 - timeout - no new data has arrived or error during copying and destination data is corrupted VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver wait in a busy polling loop until a specific STIM request is fulfilled. Depending on the STIM method used, on success all transfer descriptors assigned to stim_event are processed and the STIM data is transferred to its destination. Alternative version of VX1000If_StimWaitVoid with return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-9 VX1000If_StimWait

5.1.2.5 VX1000If_StimWaitVoid

Prototype	
void **VX1000If_StimWaitVoid** (uint8 stim_event, uint32 timeout_us)	
Parameter	
stim_event	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
timeout_us	timeout in microseconds, starting from related call to VX1000If_StimRequest
Return code	
void	-
Functional Description	
Makes the VX1000 device driver wait in a busy polling loop until a specific STIM request is fulfilled. Depending on the STIM method used, on success all transfer descriptors assigned to stim_event are processed and the STIM data is transferred to its destination. Alternative version of VX1000If_StimWait without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-10 VX1000If_StimWaitVoid

5.1.2.6 VX1000If_BypassStim

Prototype	
uint8 **VX1000If_BypassStim** (uint8 stim_event)	
Parameter	
stim_event	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
uint8	0 - data copied successfully 1 - error during copying and destination data corrupted VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver wait in a busy polling loop until a specific STIM request is fulfilled. Depending on the STIM method used, on success all transfer descriptors assigned to stim_event are processed and the STIM data is transferred to its destination. Alternative version of VX1000If_BypassStimVoid with return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-11 VX1000If_BypassStim

5.1.2.7 VX1000If_BypassStimVoid

Prototype	
`void` **`VX1000If_BypassStimVoid`** `(uint8 stim_event)`	
Parameter	
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver wait in a busy polling loop until a specific STIM request is fulfilled. Depending on the STIM method used, on success all transfer descriptors assigned to stim_event are processed and the STIM data is transferred to its destination. Alternative version of VX1000If_BypassStim without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-12 VX1000If_BypassStimVoid

5.1.2.8 VX1000If_StimActive

Prototype	
`uint8` **`VX1000If_StimActive`** `(uint8 stim_event)`	
Parameter	
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
`uint8`	1 - active 0 - inactive VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver return whether STIM is active for the specific event channel and globally.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-13 VX1000If_StimActive

5.1.2.9 VX1000If_StimSkip

Prototype
`void` **`VX1000If_StimSkip`** `(uint8 stim_event)`

Parameter	
stim_event	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
void	-
Functional Description	
Makes the VX1000 device driver send a STIM skip event to tell the attached VX1000 device not to stimulate the next cycle. This is in effect a dummy STIM request.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-14 VX1000If_StimSkip

5.1.2.10 VX1000If_Stimulate

Prototype	
uint8 **VX1000If_Stimulate** (uint8 stim_trigger_event, uint8 stim_event, uint8 cycle_delay, uint32 timeout_us)	
Parameter	
stim_trigger_event	DAQ event range
stim_event	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
cycle_delay	Specifies the number of cycles between triggering and the associated stimulation. During the first cycle_delay cycles there is no stimulation in the ECU, instead the VX1000 device fills its STIM buffer FIFO.
timeout_us	Timeout in microseconds, starting upon calling this function.
Return code	
uint8	0 - code to be bypassed shall be executed as bypassing is not active 1 - STIM successful, code to be bypassed shall be skipped 2 - STIM failed, it is up to the application to handle this error VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver perform a complete stimulation. This is done by requesting STIM data using the stim_trigger_event which is in fact a DAQ event with or without DAQ data. The first cycle_delay calls are used to fill a STIM data pipeline in the VX1000. During these calls the function will return 0. After the first cycle_delay calls, this function will busy wait with timeout for the VX1000 to actually complete the stimulation (stim_event). The pipeline depth / initial delay has to be considered when generating the STIM data. Alternative version of VX1000If_StimulateVoid with return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-15 VX1000If_Stimulate

5.1.2.11 VX1000If_StimulateVoid

Prototype	
`void VX1000If_StimulateVoid (uint8 stim_trigger_event, uint8 stim_event, uint8 cycle_delay, uint32 timeout_us)`	
Parameter	
`stim_trigger_event`	DAQ event range
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
`cycle_delay`	Specifies the number of cycles between the triggering and the associated stimulation. During first cycle_delay cycles the ECU will not be stimulated in the ECU, instead the VX1000 device will fill its STIM data pipeline.
`timeout_us`	Timeout in microseconds, starting with the function call
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver perform a complete stimulation. This is done by requesting STIM data using the stim_trigger_event which is in fact a DAQ event with or without DAQ data. The first cycle_delay calls are used to fill a STIM data pipeline in the VX1000. After the first cycle_delay calls, this function will busy wait with timeout for the VX1000 to actually complete the stimulation (stim_event). The pipeline depth / initial delay has to be considered when generating the STIM data. Alternative version of VX1000If_Stimulate without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-16 VX1000If_StimulateVoid

5.1.2.12 VX1000If_Bypass

Prototype	
`uint8 VX1000If_Bypass (uint8 daq_event, uint8 stim_event, uint32 timeout_us)`	
Parameter	
`daq_event`	DAQ event range
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
`timeout_us`	Timeout in microseconds, starting with the function call
Return code	
`uint8`	0 - code to be bypassed shall be executed as bypassing is not active 1 - Bypass successful, code to be bypassed shall be skipped 2 - Bypass failed, it is up to the application to handle this error VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE

Functional Description
Makes the VX1000 device driver initiate a bypass by sending DAQ data to the tool and implicitly requesting a stimulation (daq_event), then busy wait with timeout for the tool to complete the stimulation (stim_event). Alternative version of VX1000If_BypassVoid with return values.
Particularities and Limitations
> VX1000If_InitAsyncStart must have been called.

Table 5-17 VX1000If_Bypass

5.1.2.13 VX1000If_BypassVoid

Prototype	
`void` **`VX1000If_BypassVoid`** `(uint8 daq_event, uint8 stim_event, uint32 timeout_us)`	
Parameter	
`daq_event`	DAQ event range
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
`timeout_us`	Timeout in microseconds, starting with the function call
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver initiate a bypass by sending DAQ data to the tool and implicitly requesting a stimulation (daq_event), then busy wait with timeout for the tool to complete the stimulation (stim_event). Alternative version of VX1000If_Bypass without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-18 VX1000If_BypassVoid

5.1.2.14 VX1000If_BypassDaq

Prototype	
`uint8` **`VX1000If_BypassDaq`** `(uint8 daq_event, uint8 stim_event)`	
Parameter	
`daq_event`	DAQ event range
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
`uint8`	0 - bypassing is not active: code to be bypassed shall be executed 1 - bypassing is active: code to be bypassed shall shall be skipped VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE

Functional Description
Makes the VX1000 device driver initiate a bypass by sending a DAQ event followed by a STIM request. Alternative version of VX1000If_BypassDaqVoid with return values.
Particularities and Limitations
> VX1000If_InitAsyncStart() must have been called.

Table 5-19 VX1000If_BypassDaq

5.1.2.15 VX1000If_BypassDaqVoid

Prototype	
`void` **`VX1000If_BypassDaqVoid`** `(uint8 daq_event, uint8 stim_event)`	
Parameter	
`daq_event`	DAQ event range
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver initiate a bypass by sending a DAQ event followed by a STIM request. Alternative version of VX1000If_BypassDaq without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-20 VX1000If_BypassDaqVoid

5.1.2.16 VX1000If_BypassTrigger

Prototype	
`uint8` **`VX1000If_BypassTrigger`** `(uint8 daq_event, uint8 stim_event)`	
Parameter	
`daq_event`	DAQ event range
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
`uint8`	0 - bypassing is not active: code to be bypassed shall be executed 1 - bypassing is active: code to be bypassed shall shall be skipped VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver initiate a bypass by sending DAQ data to the tool and implicitly requesting a stimulation (daq_event). Alternative version of VX1000If_BypassTriggerVoid with return values.	

Particularities and Limitations
> VX1000If_InitAsyncStart must have been called.

Table 5-21 VX1000If_BypassTrigger

5.1.2.17 VX1000If_BypassTriggerVoid

Prototype	
`void` **`VX1000If_BypassTriggerVoid`** `(uint8 daq_event, uint8 stim_event)`	
Parameter	
`daq_event`	DAQ event range
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver initiate a bypass by sending DAQ data to the tool and implicitly requesting a stimulation (daq_event). Alternative version of VX1000If_BypassTrigger without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-22 VX1000If_BypassTriggerVoid

5.1.2.18 VX1000If_BypassWait

Prototype	
`uint8` **`VX1000If_BypassWait`** `(uint8 stim_event, uint32 timeout_us)`	
Parameter	
`stim_event`	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
`timeout_us`	Timeout in microseconds, starting with the function call
Return code	
`uint8`	0 - bypassed code shall be activated because bypassing is not active 1 - everything done, bypassed code shall be disabled 2 - bypassing failed; it's up to the application design whether executing the bypassed code makes sense here VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver perform a busy wait with timeout for the tool to complete a stimulation (stim_event) that has been initiated beforehand by an appropriate call to VX1000If_BypassTrigger. Alternative version of VX1000If_BypassWaitVoid with return values.	

Particularities and Limitations
> **VX1000If**_InitAsyncStart must have been called.

Table 5-23 VX1000If_BypassWait

5.1.2.19 VX1000If_BypassWaitVoid

Prototype	
void **VX1000If_BypassWaitVoid** (uint8 stim_event, uint32 timeout_us)	
Parameter	
stim_event	The range for stim_event is defined in the VX1000 device driver configuration. It is a subset of the available DAQ event channels.
timeout_us	Timeout in microseconds, starting with the function call
Return code	
void	-
Functional Description	
Makes the VX1000 device driver perform a busy wait with timeout for the tool to complete a stimulation (stim_event) that has been initiated beforehand by an appropriate call to VX1000If_BypassTrigger. Alternative version of VX1000If_BypassWait without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-24 VX1000If_BypassWaitVoid

5.1.2.20 VX1000If_Event

Prototype	
void **VX1000If_Event** (uint8 eventNumber)	
Parameter	
eventNumber	DAQ event range
Return code	
void	-
Functional Description	
Makes the VX1000 device driver trigger an XCP event. For copying-mechanism-based DAQ, makes the VX1000 device driver process all transfer descriptors assigned to eventNumber and to copy the DAQ data to an intermediate buffer to be read by the VX1000.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-25 VX1000If_Event

5.1.3 Generic Hooking Control

5.1.3.1 VX1000If_HookTrigger

Prototype	
uint8 **VX1000If_HookTrigger** (uint16 hook_id)	
Parameter	
hook_id	Hook id range
Return code	
uint8	0 - inactive bypass or active bypass and original code enabled 1 - bypass active and original code disabled VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver trigger a generic bypass whose event IDs are related to HookID. Alternative version of VX1000If_HookTriggerVoid with return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-26 VX1000If_HookTrigger

5.1.3.2 VX1000If_HookTriggerVoid

Prototype	
void **VX1000If_HookTriggerVoid** (uint16 hook_id)	
Parameter	
hook_id	Hook id range
Return code	
void	-
Functional Description	
Makes the VX1000 device driver trigger a generic bypass whose event IDs are related to HookID. Alternative version of VX1000If_HookTrigger without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-27 VX1000If_HookTriggerVoid

5.1.3.3 VX1000If_HookWait

Prototype	
uint8 **VX1000If_HookWait** (uint16 hook_id, uint32 timeout_us)	
Parameter	
hook_id	Hook id range

`timeout_us`	Timeout in microseconds, starting right now.
Return code	
`uint8`	0 - bypass inactive 1 - stimulation done, no timeout, OK 2 - stimulation not done, timeout 3 - stimulation not done, timeout, execute original code VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver stimulate with timeout for a generic bypass whose event IDs are related to HookID. Optionally, an additional DAQ event will be triggered. Alternative version of VX1000If_HookWaitVoid with return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-28 VX1000If_HookWait

5.1.3.4 VX1000If_HookWaitVoid

Prototype	
`void` **`VX1000If_HookWaitVoid`** `(uint16 hook_id, uint32 timeout_us)`	
Parameter	
`hook_id`	Hook id range
`timeout_us`	Timeout in microseconds, starting with the function call
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver stimulate with timeout for a generic bypass whose event IDs are related to HookID. Optionally, an additional DAQ event will be triggered. Alternative version of VX1000If_HookWait without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-29 VX1000If_HookWaitVoid

5.1.3.5 VX1000If_GenericEvent

Prototype	
`void` **`VX1000If_GenericEvent`** `(uint16 hook_id)`	
Parameter	
`hook_id`	Hook id range
Return code	
`void`	-

Functional Description
Makes the VX1000 device driver trigger a generic event whose event ID is related to hook_id.
Particularities and Limitations
> VX1000If_InitAsyncStart must have been called.

Table 5-30 VX1000If_GenericEvent

5.1.3.6 VX1000If_Hook

Prototype	
void **VX1000If_Hook** (uint16 hook_id, uint32 timeout_us, code)	
Parameter	
hook_id	Hook id range as configured in the VX1000 device driver
timeout_us	Timeout in microseconds, starting with the function call
code	User code to be executed in case of failed stimulation or inactive hook.
Return code	
void	-
Functional Description	
Makes the VX1000 device driver trigger a generic bypass for STIM events with VX1000 Hooks, then wait until the data set requests of this generic bypass are finished successfully and then trigger a DAQ event or in case of inactive hook or failed stimulation to execute user code without triggering the DAQ event.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-31 VX1000If_Hook

5.1.4 Hook Based Bypassing

5.1.4.1 VX1000If_BypassHbbGetval8

Prototype	
uint8 **VX1000If_BypassHbbGetval8** (uint16 hook_id, uint8 default)	
Parameter	
hook_id	HBB id range as configured in the VX1000 device driver
default	Specifies the default value to be returned if hook is not valid.
Return code	
uint8	Data corresponding to the stimulated value, if hook is valid, data is available and VX1000If_ returns TRUE. Otherwise default is returned.
Functional Description	
Makes the VX1000 device driver check whether valid data corresponding to the given Hook ID is present in the buffer.	

Particularities and Limitations
> VX1000If_InitAsyncStart must have been called.

Table 5-32 VX1000If_BypassHbbGetval8

5.1.4.2 VX1000If_BypassHbbGetval16

Prototype	
`uint16 VX1000If_BypassHbbGetval16 (uint16 hook_id, uint16 default)`	
Parameter	
`hook_id`	HBB id range as configured in the VX1000 device driver
`default`	Specifies the default value to be returned if hook is not valid.
Return code	
`uint16`	Data corresponding to the stimulated value, if hook is valid, data is available and VX1000If_ returns TRUE. Otherwise default is returned.
Functional Description	
Makes the VX1000 device driver check whether valid data corresponding to the given Hook ID is present in the buffer.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-33 VX1000If_BypassHbbGetval16

5.1.4.3 VX1000If_BypassHbbGetval32

Prototype	
`uint32 VX1000If_BypassHbbGetval32 (uint16 hook_id, uint32 default)`	
Parameter	
`hook_id`	HBB id range as configured in the VX1000 device driver
`default`	Specifies the default value to be returned if hook is not valid.
Return code	
`uint32`	Data corresponding to the stimulated value, if hook is valid, data is available and VX1000If_ returns TRUE. Otherwise default is returned.
Functional Description	
Makes the VX1000 device driver check whether valid data corresponding to the given Hook ID is present in the buffer.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-34 VX1000If_BypassHbbGetval32

5.1.4.4 VX1000If_BypassHbbGetval64

Prototype	
`uint64` **`VX1000If_BypassHbbGetval64`** `(uint16 hook_id, uint64 default)`	
Parameter	
`hook_id`	HBB id range as configured in the VX1000 device driver
`default`	Specifies the default value to be returned if hook is not valid.
Return code	
`uint64`	Data corresponding to the stimulated value, if hook is valid, data is available and VX1000If_ returns TRUE. Otherwise default is returned.
Functional Description	
Makes the VX1000 device driver check whether valid data corresponding to the given Hook ID is present in the buffer.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-35 VX1000If_BypassHbbGetval64

5.1.4.5 VX1000If_BypassHbbGetvalFloat

Prototype	
`float32` **`VX1000If_BypassHbbGetvalFloat`** `(uint16 hook_id, float32 default)`	
Parameter	
`hook_id`	HBB id range as configured in the VX1000 device driver
`default`	Specifies the default value to be returned if hook is not valid.
Return code	
`float32`	Data corresponding to the stimulated value, if hook is valid, data is available and VX1000If_ returns TRUE. Otherwise default is returned.
Functional Description	
Makes the VX1000 device driver check whether valid data corresponding to the given Hook ID is present in the buffer.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-36 VX1000If_BypassHbbGetvalFloat

5.1.4.6 VX1000If_BypassHbbGetvalDouble

Prototype
`float64` **`VX1000If_BypassHbbGetvalDouble`** `(uint16 hook_id, float64 default)`

Parameter	
`hook_id`	HBB id range as configured in the VX1000 device driver
`default`	Specifies the default value to be returned if hook is not valid.
Return code	
`float64`	Data corresponding to the stimulated value, if hook is valid, data is available and VX1000If_ returns TRUE. Otherwise default is returned.
Functional Description	
Makes the VX1000 device driver check whether valid data corresponding to the given Hook ID is present in the buffer.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-37 VX1000If_BypassHbbGetvalDouble

5.1.5 Mailbox

5.1.5.1 VX1000If_MailboxControl

Prototype	
`void` **`VX1000If_MailboxControl`** `(void)`	
Parameter	
`void`	-
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver check the VX1000 mailbox for pending requests and trigger necessary actions.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function.	

Table 5-38 VX1000If_MailboxControl

5.1.5.2 VX1000If_MailboxWrite

Prototype	
`uint32` **`VX1000If_MailboxWrite`** `(uint16 len, const uint8* pBuf)`	
Parameter	
`len`	Speficies the message size in bytes.
`pBuf`	Specifies the pointer to message data input.

Return code	
`uint32`	VX1000_MAILBOX_OK – mailbox transfer successful VX1000_MAILBOX_ERR_FULL - error: no free mailbox slots available VX1000_MAILBOX_ERR_NULL - error: pBuf is null pointer VX1000_MAILBOX_ERR_SIZE - error: len exceeds mailbox slot size VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver write len bytes from pBuf to the Slave->Master mailbox and notify the master. Alternative version of VX1000If_MailboxWriteVoid with return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function. > Some of the return values are defined in the VX1000 device driver.	

Table 5-39 VX1000If_MailboxWrite

5.1.5.3 VX1000If_MailboxWriteVoid

Prototype	
`void` **`VX1000If_MailboxWriteVoid`** `(uint16 len, const uint8* pBuf)`	
Parameter	
`len`	Speficies the message size in bytes.
`pBuf`	Specifies the pointer to message data input.
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver write len bytes from pBuf to the Slave->Master mailbox and notify the master. Alternative version of VX1000If_MailboxWrite without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function.	

Table 5-40 VX1000If_MailboxWriteVoid

5.1.5.4 VX1000If_MailboxWriteSplit

Prototype
`uint32` **`VX1000If_MailboxWriteSplit`** `(uint32** ppBuf)`

Parameter	
ppBuf	(IN): pointer to a pointer variable. (*OUT): pointer to the data field of the next free Slave->Master mailbox.
Return code	
uint32	VX1000_MAILBOX_OK – split write transaction successfully initiated VX1000_MAILBOX_ERR_FULL - error: no free mailbox slots available VX1000_MAILBOX_ERR_NULL - error: ppBuf is a null pointer VX1000_MAILBOX_ERR_SPLIT_PEND - error: another split mailbox write transaction is pending VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver find out the location of the next unused message buffer and return the info to the caller. The mailbox state is neither changed nor is the master notified. To finalize writing data to the mailbox, VX1000If_MailboxWriteDone must be called. Alternative version of VX1000If_MailboxWriteSplitVoid with return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function. > Some of the return values are defined in the VX1000 device driver.	

Table 5-41 VX1000If_MailboxWriteSplit

5.1.5.5 VX1000If_MailboxWriteSplitVoid

Prototype	
`void VX1000If_MailboxWriteSplitVoid (uint32** ppBuf)`	
Parameter	
ppBuf	(IN): pointer to a pointer variable. (*OUT): pointer to the data field of the next free Slave->Master mailbox.
Return code	
void	-
Functional Description	
Makes the VX1000 device driver find out the location of the next unused message buffer and return the info to the caller. The mailbox state is neither changed nor is the master notified. To finalize writing data to the mailbox, VX1000If_MailboxWriteDone must be called. Alternative version of VX1000If_MailboxWriteSplit without return values.	

Particularities and Limitations
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function.

Table 5-42 VX1000If_MailboxWriteSplitVoid

5.1.5.6 VX1000If_MailboxWriteDone

Prototype	
uint32 **VX1000If_MailboxWriteDone** (uint32 len)	
Parameter	
len	The size of the entire message in bytes.
Return code	
uint32	VX1000_MAILBOX_OK – mailbox split write transaction completed VX1000_MAILBOX_ERR_SIZE - error: len exceeds mailbox slot size VX1000_MAILBOX_ERR_SPLIT_PEND - error: no pending write transaction VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver finalize a Slave->Master mailbox transfer that has been started by calling VX1000If_MailboxWriteSplit. Alternative version of VX1000If_MailboxWriteDoneVoid with return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart and VX1000If_MailboxWriteSplit must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function. > Some of the return values are defined in the VX1000 device driver.	

Table 5-43 VX1000If_MailboxWriteDone

5.1.5.7 VX1000If_MailboxWriteDoneVoid

Prototype	
void **VX1000If_MailboxWriteDoneVoid** (uint32 len)	
Parameter	
len	The size of the entire message in bytes.
Return code	
void	-
Functional Description	
Makes the VX1000 device driver finalize a Slave->Master mailbox transfer that has been started by calling VX1000If_MailboxWriteSplit. Alternative version of VX1000If_MailboxWriteDone without return values.	

Particularities and Limitations
> VX1000If_InitAsyncStart and VX1000If_MailboxWriteSplit must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function.

Table 5-44 VX1000If_MailboxWriteDoneVoid

5.1.5.8 VX1000If_MailboxRead

Prototype	
`uint32 VX1000If_MailboxRead (uint32* pLen, uint8* pBuf)`	
Parameter	
`pLen`	Pointer holding the maximum allowed message size. The value is overwritten with the actual message size if successful.
`pBuf`	Pointer to destination for the next message. The caller is responsible that the destination contains at least *pLen writeable bytes. The function aborts with an error if the buffer is too small for the current message (no bytes copied).
Return code	
`uint32`	VX1000_MAILBOX_OK – mailbox transfer successful VX1000_MAILBOX_ERR_EMPTY - error: mailbox is empty VX1000_MAILBOX_ERR_NULL - error: pLen or pBuf are null pointers VX1000_MAILBOX_ERR_SIZE - error: mailbox slot content exceeds pLen VX1000_MAILBOX_ERR_SPLIT_PEND - error: split read transaction pending VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver read the data from the next filled Master->Slave mailbox slot into pBuf and return the number of bytes in pLen. Alternative version of VX1000If_MailboxReadVoid with return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function. > Some of the return values are defined in the VX1000 device driver.	

Table 5-45 VX1000If_MailboxRead

5.1.5.9 VX1000If_MailboxReadVoid

Prototype
`void VX1000If_MailboxReadVoid (uint32* pLen, uint8* pBuf)`

Parameter	
pLen	Pointer holding the maximum allowed message size. The value is overwritten with the actual message size if successful.
pBuf	Pointer to destination for the next message. The caller is responsible that the destination contains at least *pLen writeable bytes. The function aborts with an error if the buffer is too small for the current message (no bytes copied).
Return code	
void	-
Functional Description	
Makes the VX1000 device driver read the data from the next filled Master->Slave mailbox slot into pBuf and return the number of bytes in pLen. Alternative version of VX1000If_MailboxRead without return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function.	

Table 5-46 VX1000If_MailboxReadVoid

5.1.5.10 VX1000If_MailboxReadSplit

Prototype	
uint32 **VX1000If_MailboxReadSplit** (uint32* pLen, uint32** ppBuf)	
Parameter	
pLen	Pointer to a 32bit variable. The value is overwritten with the byte count of the next message if successful. The caller is responsible that the pointer is valid and that the destination is writeable
ppBuf	Pointer to the data field of the next unread message. The caller is responsible that the pointer is valid and that the destination is writeable.
Return code	
uint32	VX1000_MAILBOX_OK – mailbox transfer successful VX1000_MAILBOX_ERR_EMPTY - error: Mailbox is empty VX1000_MAILBOX_ERR_NULL - pLen or ppBuf is a null pointer VX1000_MAILBOX_ERR_SPLIT_PEND - another split read transaction is pending VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE

Functional Description
Makes the VX1000 device driver return the location and length of the next unread mailbox message. Note: the mailbox state is not changed nor is the master notified. VX1000If_MAILBOX_READDONE must be called to complete the transaction. Alternative version of VX1000If_MailboxReadSplitVoid with return value.
Particularities and Limitations
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function. > Some of the return values are defined in the VX1000 device driver.

Table 5-47 VX1000If_MailboxReadSplit

5.1.5.11 VX1000If_MailboxReadSplitVoid

Prototype	
`void` **`VX1000If_MailboxReadSplitVoid`** `(uint32* pLen, uint32** ppBuf)`	
Parameter	
`pLen`	Pointer to a 32bit variable. The value is overwritten with the byte count of the next message if successful. The caller is responsible that the pointer is valid and that the destination is writeable
`ppBuf`	Pointer to the data field of the next unread message. The caller is responsible that the pointer is valid and that the destination is writeable.
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver return the location and length of the next unread mailbox message. Note: the mailbox state is not changed nor is the master notified. VX1000If_MAILBOX_READDONE must be called to complete the transaction. Alternative version of VX1000If_MailboxReadSplit without return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function.	

Table 5-48 VX1000If_MailboxReadSplitVoid

5.1.5.12 VX1000If_MailboxReadDone

Prototype
`uint32` **`VX1000If_MailboxReadDone`** `(void)`

Parameter	
`void`	-
Return code	
`uint32`	VX1000_MAILBOX_OK – mailbox transfer successful VX1000_MAILBOX_ERR_SPLIT_PEND - no pending read split transaction VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver mark the Master->Slave mailbox slot for the pending read transaction as empty and notify the master afterwards. Alternative version of VX1000If_MailboxReadDoneVoid with return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart and VX1000If_MailboxReadSplit must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function. > Some of the return values are defined in the VX1000 device driver.	

Table 5-49 VX1000If_MailboxReadDone

5.1.5.13 VX1000If_MailboxReadDoneVoid

Prototype	
`void` **`VX1000If_MailboxReadDoneVoid`** `(void)`	
Parameter	
`void`	-
Return code	
`void`	-
Functional Description	
Makes the VX1000 device driver mark the Master->Slave mailbox slot for the pending read transaction as empty and notify the master afterwards. Alternative version of VX1000If_MAILBOX_READDONE without return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart and VX1000If_MailboxReadSplit must have been called. > This function must not be interrupted by any VX1000 mailbox write function. > This function must not interrupt any VX1000 mailbox write function.	

Table 5-50 VX1000If_MailboxReadDoneVoid

5.1.6 Overlay

5.1.6.1 VX1000If_OvlSetConfig

Prototype
`uint8` **`VX1000If_OvlSetConfig`** `(uint32 value, uint32 mask, uint8 page, uint32 master, uint32 calMaster)`

Parameter	
value	Overlay windows to be activated/deactivated.
mask	Resource Mask
page	Overlay Page
master	Masters to be activated
calMaster	Masters resource Mask
Return code	
uint8	0 - Nothing done 1 - Page switch done 2 - Value not written correctly 3 - No single-master page-switch possible 4 - Generic error VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 device driver execute a derivative-specific method to globally enable/disable overlays. Note: the VX1000 device driver assumes exclusive ownership of the overlay unit. Alternative version of VX1000If_OvlSetConfigVoid with return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-51 VX1000If_OvlSetConfig

5.1.6.2 VX1000If_OvlSetConfigVoid

Prototype	
void **VX1000If_OvlSetConfigVoid** (uint32 value, uint32 mask, uint8 page, uint32 master, uint32 calMaster)	
Parameter	
value	Overlay windows to be activated/deactivated.
mask	Resource Mask
page	Overlay Page
master	Masters to be activated
calMaster	Masters resource Mask
Return code	
void	-
Functional Description	
Makes the VX1000 device driver execute a derivative-specific method to globally enable/disable overlays. Note: the VX1000 device driver assumes exclusive ownership of the overlay unit. Alternative version of VX1000If_OvlSetConfig without return value.	

Particularities and Limitations
> VX1000If_InitAsyncStart must have been called.

Table 5-52 VX1000If_OvlSetConfigVoid

5.1.6.3 VX1000If_OvlSetConfigDone

Prototype	
uint8 **VX1000If_OvlSetConfigDone** (uint32 value, uint32 mask, uint8 page, uint32 master, uint32 calMaster)	
Parameter	
value	Overlay windows to be activated/deactivated.
mask	Resource Mask
page	Overlay Page
master	Masters to be activated
calMaster	Masters resource Mask
Return code	
uint8	0 - Nothing done 1 - Page switch done 2 - Value not written correctly 3 - No single-master page-switch possible 4 - Generic error VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Transmits the status of a page switching attempt to the VX1000 driver and optionally also to the XCP tool. Alternative version of VX1000If_OvlSetConfigDoneVoid with return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-53 VX1000If_OvlSetConfigDone

5.1.6.4 VX1000If_OvlSetConfigDoneVoid

Prototype	
void **VX1000If_OvlSetConfigDoneVoid** (uint32 value, uint32 mask, uint8 page, uint32 master, uint32 calMaster)	
Parameter	
value	Overlay windows to be activated/deactivated.
mask	Resource Mask
page	Overlay Page
master	Masters to be activated
calMaster	Masters resource Mask

Return code	
`void`	-
Functional Description	
Transmits the status of a page switching attempt to the VX1000 driver and optionally also to the XCP tool. Alternative version of VX1000If_OvlSetConfigDone without return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-54 VX1000If_OvlSetConfigDoneVoid

5.1.6.5 VX1000If_OvlChkPageSwDone

Prototype	
`uint8` **`VX1000If_OvlChkPageSwDone`** `(void)`	
Parameter	
`void`	-
Return code	
`uint8`	0 - Nothing done 1 - Page switch done 2 - Value not written correctly 3 - No single-master page-switch possible 4 - Generic error VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 driver check and finalize page switching status of all bus masters for which a page switch was requested. Alternative version of VX1000If_OvlChkPageSwDoneVoid with return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-55 VX1000If_OvlChkPageSwDone

5.1.6.6 VX1000If_OvlChkPageSwDoneVoid

Prototype	
`void` **`VX1000If_OvlChkPageSwDoneVoid`** `(void)`	
Parameter	
`void`	-
Return code	
`void`	-

Functional Description
Makes the VX1000 driver check and finalize page switching status of all bus masters for which a page switch was requested. Alternative version of VX1000If_OvlChkPageSwDone without return value.
Particularities and Limitations
> VX1000If_InitAsyncStart must have been called.

Table 5-56 VX1000If_OvlChkPageSwDoneVoid

5.1.6.7 VX1000If_OvlChkPageSwCore

Prototype	
`uint8 VX1000If_OVL_CHK_PAGESW_CORE (uint32 master)`	
Parameter	
`master`	Bus master to be checked
Return code	
`uint8`	0 - Nothing done 1 - Page switch done 2 - Value not written correctly 3 - No single-master page-switch possible 4 - Generic error VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 driver check the page switching status of a specific bus master. Alternative version of VX1000If_OvlChkPageSwCoreVoid with return value.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-57 VX1000If_OvlChkPageSwCore

5.1.6.8 VX1000If_OvlChkPageSwCoreVoid

Prototype	
`void VX1000If_OvlChkPageSwCoreVoid (uint32 master)`	
Parameter	
`master`	Bus master to be checked
Return code	
`void`	-
Functional Description	
Makes the VX1000 driver check the page switching status of a specific bus master. Alternative version of VX1000If_OvlChkPageSwCore without return value.	

Particularities and Limitations
> VX1000If_InitAsyncStart must have been called.

Table 5-58 VX1000If_OvlChkPageSwCoreVoid

5.1.6.9 VX1000If_OvlIsPageSwRequested

Prototype	
uint8 **VX1000If_OvlIsPageSwRequested** (uint32 master)	
Parameter	
master	Bus master to be checked
Return code	
uint8	0 - Page switch is not pending 1 - Page switch is pending VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 driver check whether a page switch request is pending for a bus master or not.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-59 VX1000If_OvlIsPageSwRequested

5.1.6.10 VX1000If_InvalidateEmem

Prototype	
void **VX1000If_InvalidateEmem** (void)	
Parameter	
void	-
Return code	
void	-
Functional Description	
Makes the VX1000 driver invalidate the signature of the VX1000-allocated persistent ECU-RAM.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-60 VX1000If_InvalidateEmem

5.1.6.11 VX1000If_CalWakeupRequested

Prototype	
uint8 **VX1000If_CalWakeupRequested** (void)	
Parameter	
void	-

Return code	
uint8	0 - No Calibration Wakeup request pending 1 - Calibration Wakeup request pending VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 driver check whether the XCP tool has requested a wakeup for calibration purposes.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-61 VX1000If_CalWakeupRequested

5.1.6.12 VX1000If_IsCalWakeupActive

Prototype	
uint8 **VX1000If_IsCalWakeupActive** (void)	
Parameter	
void	-
Return code	
uint8	0 - ECU need not stay awake 1 - ECU must stay awake VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 driver check whether the ECU must stay awake for calibration purposes or not.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-62 VX1000If_IsCalWakeupActive

5.1.7 Resource Management

5.1.7.1 VX1000If_EnableAccess

Prototype	
void **VX1000If_EnableAccess** (void)	
Parameter	
void	-
Return code	
void	-
Functional Description	
Makes the VX1000 driver enable the VX1000 tool access.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-63 VX1000If_EnableAccess

5.1.7.2 VX1000If_DisableAccess

Prototype	
uint8 **VX1000If_DisableAccess** (void)	
Parameter	
void	-
Return code	
uint8	0 – VX1000 tool access successfully disabled 1 – unable to disable VX1000 tool access VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 driver disable the VX1000 tool access. Alternative version of VX1000If_DisableAccessVoid with return values.	
Particularities and Limitations	
> **VX1000If_**InitAsyncStart must have been called.	

Table 5-64 VX1000If_DisableAccess

5.1.7.3 VX1000If_DisableAccessVoid

Prototype	
void **VX1000If_DisableAccessVoid** (void)	
Parameter	
void	-
Return code	
void	-
Functional Description	
Makes the VX1000 driver disable the VX1000 tool access. Alternative version of VX1000If_DisableAccess without return values.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-65 VX1000If_DisableAccessVoid

5.1.7.4 VX1000If_IsAccessDisabled

Prototype	
boolean **VX1000If_IsAccessDisabled** (void)	
Parameter	
void	-

Return code	
boolean	TRUE - VX1000 tool access disabled FALSE - VX1000 tool access enabled VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 driver check whether the VX1000 tool access is disabled or not.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-66 VX1000If_IsAccessDisabled

5.1.8 User functions

5.1.8.1 VX1000If_DetectFklRequests

Prototype	
void **VX1000If_DetectFklRequests** (void)	
Parameter	
void	-
Return code	
void	-
Functional Description	
Makes the VX1000 driver prevent the application from writing to RAM to allow a flash kernel download by the VX1000 and to busily wait for trigger command to jump to the flash kernel execution start address provided by the VX1000.	
Particularities and Limitations	
> VX1000If_InitAsyncStart must have been called.	

Table 5-67 VX1000If_DetectFklRequests

5.1.8.2 VX1000If_DeviceDetected

Prototype	
uint8 **VX1000If_DeviceDetected** (void)	
Parameter	
void	-
Return code	
uint8	0 - not detected 1 – detected VX1000IF_RET_E_NOT_OK - while VX1000If_ returned FALSE
Functional Description	
Makes the VX1000 driver check whether a VX1000 has been detected.	

Particularities and Limitations
> None

Table 5-68 VX1000If_DeviceDetected

5.1.8.3 VX1000If_DynAddrSig_UpdateAddress

Prototype
VX1000If_DynAddrSig_UpdateAddress
Functional Description
This API function is not described here. The appropriate VX1000 AppDriver Addons are documented in Technical Reference Supplement VX1000If [4].

Table 5-69 VX1000If_DynAddSig_UpdateAddress

5.2 Callout Macros

The VX1000If interface defines callout macros. The declarations of the callout macros are provided by the BSW module, i.e. the VX1000If. It is the integrator's task to provide the corresponding macro definitions. The definitions of the callouts can be adjusted to the system's needs. The VX1000If macros are described in the following tables:

5.2.1 VX1000If_IsVX1000DriverAccessEnabled

Prototype	
boolean **VX1000If_IsVX1000DriverAccessEnabled** (void)	
Parameter	
void	-
Return code	
boolean	TRUE – VX1000If forwards API calls to the VX1000 driver. Must only be used in development environments. FALSE – VX1000If component blocks all API calls towards VX1000 driver. This shall be the return value for serial production usage.
Functional Description	
The VX1000If calls this callout macro in every VX1000If service to check whether the VX1000If component is active or not. If the application returned that the VX1000If state shall be inactive the VX1000If component does not call any VX1000 device driver functions.	
Particularities and Limitations	
> This function is synchronous.	

Table 5-70 VX1000If_IsVX1000DriverAccessEnabled

Caution

As the VX1000 driver must not be accessed in serial production FALSE must be returned at any time.

If the device runs in a development environment and the VX1000 hardware shall be used, TRUE can be returned.

5.3 Services used by VX1000If

In the following table services provided by other components, which are used by VX1000If are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API

VX1000	`VX1000_INIT_ASYNC_START`
	`VX1000_INIT_ASYNC_END`
	`VX1000_PREPARE_SOFTRESET`
	`VX1000_PREPARE_SOFTRESET_VOID`
	
	`VX1000_STIM_CONTROL`
	`VX1000_BYPASS_CONTROL`
	`VX1000_STIM_REQUEST`
	`VX1000_STIM_WAIT`
	`VX1000_STIM_WAIT_VOID`
	`VX1000_BYPASS_STIM`
	`VX1000_BYPASS_STIM_VOID`
	`VX1000_STIM_ACTIVE`
	`VX1000_STIM_SKIP`
	`VX1000_STIMULATE`
	`VX1000_STIMULATE_VOID`
	`VX1000_BYPASS`
	`VX1000_BYPASS_VOID`
	`VX1000_BYPASS_DAQ`
	`VX1000_BYPASS_DAQ_VOID`
	`VX1000_BYPASS_TRIGGER`
	`VX1000_BYPASS_TRIGGER_VOID`
	`VX1000_BYPASS_WAIT`
	`VX1000_BYPASS_WAIT_VOID`
	`VX1000_EVENT`
	
	`VX1000_DETECTED`
	
	`VX1000_HOOK_TRIGGER`
	`VX1000_HOOK_TRIGGER_VOID`
	`VX1000_HOOK_WAIT`
	`VX1000_HOOK_WAIT_VOID`
	`VX1000_GENERIC_EVENT`

Component	API
	VX1000_HOOK VX1000_BYPASS_HBB_GETVAL_8 VX1000_BYPASS_HBB_GETVAL_16 VX1000_BYPASS_HBB_GETVAL_32 VX1000_BYPASS_HBB_GETVAL_64 VX1000_BYPASS_HBB_GETVAL_FLOAT VX1000_BYPASS_HBB_GETVAL_DOUBLE VX1000_MAILBOX_CONTROL VX1000_MAILBOX_WRITE VX1000_MAILBOX_WRITE_VOID VX1000_MAILBOX_WRITESPLIT VX1000_MAILBOX_WRITESPLIT_VOID VX1000_MAILBOX_WRITEDONE VX1000_MAILBOX_WRITEDONE_VOID VX1000_MAILBOX_READ VX1000_MAILBOX_READ_VOID VX1000_MAILBOX_READSPLIT VX1000_MAILBOX_READSPLIT_VOID VX1000_MAILBOX_READDONE VX1000_MAILBOX_READDONE_VOID VX1000_OVL_SET_CONFIG VX1000_OVL_SET_CONFIG_VOID VX1000_OVL_SET_CONFIG_DONE VX1000_OVL_SET_CONFIG_DONE_VOID VX1000_OVL_CHK_PAGESW_DONE VX1000_OVL_CHK_PAGESW_DONE_VOID VX1000_OVL_CHK_PAGESW_CORE VX1000_OVL_CHK_PAGESW_CORE_VOID VX1000_OVL_IS_PAGESW_REQUESTED VX1000_INVALIDATE_EMEM VX1000_CAL_WAKEUP_REQUESTED VX1000_IS_CAL_WAKEUP_ACTIVE VX1000_ENABLE_ACCESS VX1000_DISABLE_ACCESS VX1000_DISABLE_ACCESS_VOID VX1000_IS_ACCESS_DISABLED VX1000_DETECT_FKL_REQUESTS VX1000_DYNADDRSIG_UPDATEADDRESS

Table 5-71 Services used by the VX1000If

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
VX1000	The VX1000 System is a scalable solution with high performance for measurement and calibration tasks. It can be used in the vehicle – both in the interior and in the engine compartment – on test benches and in the laboratory. The system forms the interface between the ECU and a measurement and calibration tool such as CANape. For high data throughput with minimal impact on ECU run-time, data is accessed over the microcontroller-specific data trace and debug ports.
Synchronous Data Acquisition	In this mode, the MCS configures tables of memory addresses in the XCP Protocol Layer. These tables contain pointers to measurement objects, which have been configured previously for the measurement in the MCS. Each configured table is assigned to an event channel. The Xcp_Event has to be triggered cyclically for each event channel. The application has to ensure that Xcp_Event is called with the correct cycle time, which is defined in the MCS. The ECU automatically transmits the current value of the measurement objects via messages to the MCS, when the Xcp_Event is executed in the ECU's code. This means that the data can be transmitted at any particular point of the ECU code when the data values are valid.
Synchronous Data Stimulation	Synchronous Data Stimulation is the inverse mode of Synchronous Data Acquisition. The STIM processor buffers incoming data stimulation packets. When an event occurs (Xcp_Event is called), which triggers a DAQ list in data stimulation mode, the buffered data is transferred to the slave device's memory.
Bypassing	Bypassing can be realized by making use of Synchronous Data Acquisition (DAQ) and Synchronous Data Stimulation (STIM) simultaneously. State-of-the-art Bypassing also requires the administration of the bypassed functions. This administration has to be performed in a MCS like e.g. CANape. Also the slave should perform plausibility checks on the data it receives through data stimulation. The borders and actions of these checks are set by standard calibration methods.

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
ASAM	Association for Standardization of Automation and Measuring Systems
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CANape	Calibration and Measurement Data Acquisition for Electronic Control Systems
DAQ	Synchronous Data Acquistion
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MCS	Master Calibration System
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
OLDA	Online Data Aquisition
SRS	Software Requirement Specification
STIM	Synchronous Data Stimulation
SWS	Software Specification
XCP	Universal Measurement and Calibration Protocol

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com