SIEMENS

SIMATIC

V1SL

User Description

Firmware for Siemens ASIC DPC31
DPV1

Version: 1.4
Date: 04/09

Liability Exclusion
We have checked the content of this document regarding agreement with the hardware and software described. Nevertheless, deviations can't be ruled out, and we are not guaranteeing complete agreement. However, the data in this document is checked periodically. Necessary changes will be included in subsequent editions. Any suggestions for improvement are welcome.

Subject to technical changes.

Table of Contents

List of Figures

1 Introduction

Since the PROFIBUS DPV1 specification [2] was issued, essential extensions have been specified for the existing PROFIBUS DP mechanisms according to EN 50170 [1]. These are:

- Extensions to the parameterization telegram
- Extensions to the diagnostic telegram
- Additional asynchronous communication paths between Class1 master (C1 or parameterization master) and slaves:
  - Read data set (MSAC1_Read)
  - Write data set (MSAC1_Write)
  - Alarm acknowledgements (MSAC1_Alarm_Ack)

- Additional asynchronous communication paths between Class2 master (C2 master) and slaves:
  - Establishment of connection (MSAC2_Initiate)
  - Shut down of connection (MSAC2_Abort)
  - Read data set (MSAC2_Read)
  - Write data set (MSAC2_Write)
  - General data transport (MSAC2_Data_Transport)

The topic of this document is the description of the DPV1 slave firmware package (V1SL) that covers the complete functionality of a DPV1 slave according to [2].

The interfaces to the application and to the system are laid out in a way that the implementation effort for the user is low, and universal usability is ensured at the same time.

The implementation of the DPV1 slave firmware takes portability to different processors into account.

The following notes provide information regarding the use of this document. Since the firmware package and the documentation are very detailed and covers a wide variety of materials, a careful selection of the chapters to study is recommended.

To get going quickly, read the document 'V1SL Getting Started' along with the included sample application.

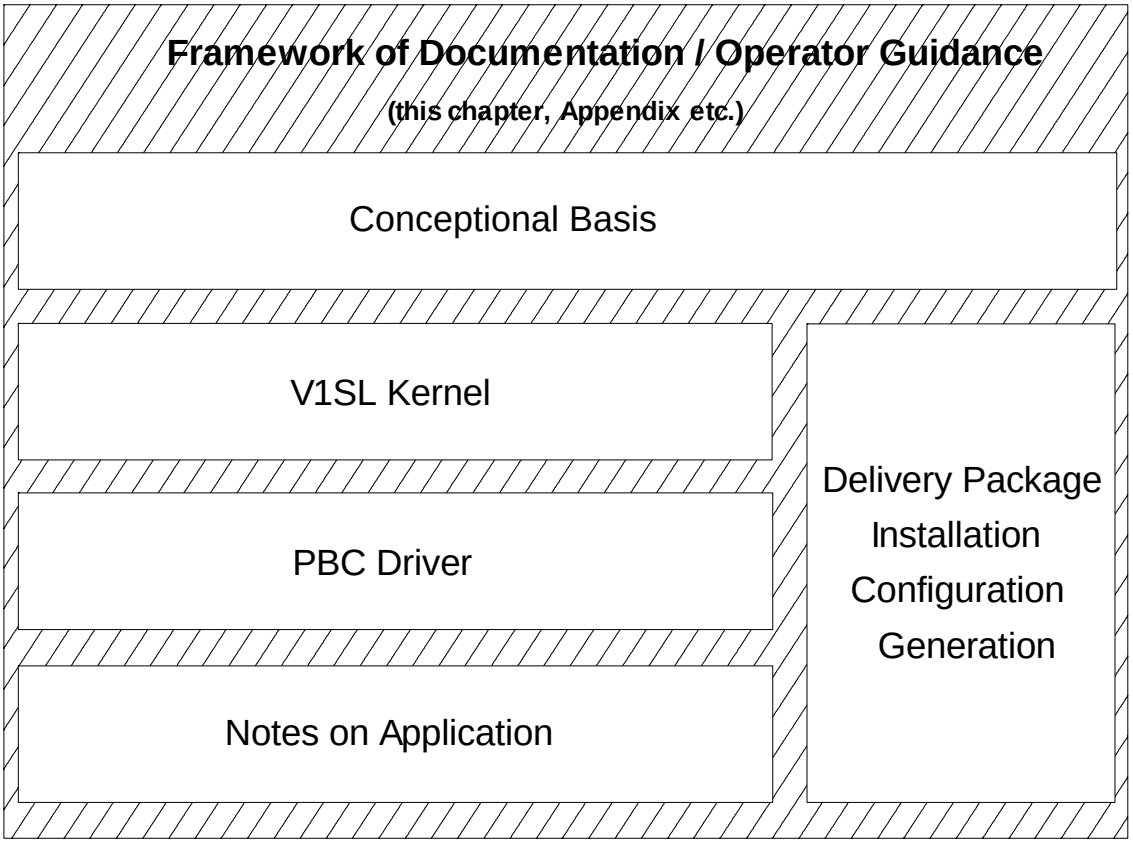


Figure 1: Structure of the V1SL Documentation

The components shown in Figure 1 allow a systematic way to become familiar with the firmware package by the following steps:

- Specify the needed firmware components by starting with the V1SL Kernel, via the PBC Driver up to the application example, which is the starting point for the DPV1 slave functionality (refer to 'V1SL Getting Started').
- Become familiar with the concept of the V1SL firmware parts, how they work and how they interface with the user or other parts of the firmware.
- Install the selected firmware components.
- Configure the selected firmware components.
- Generate the firmware.

To start processing the actions described above, it is suggested to continue with Chapter 2 'Overview' on page 10.

2 Overview

2.1 Features and Positioning of the V1SL Firmware Package

The V1SL provides the user with a powerful firmware package that can be applied universally. It has the following features:

Functionality:

- Implementation of the DP Standard Slave functionality according to [1].
- Implementation of the DPV1 slave functionality according to [2].
- The functionality of the firmware package can be scaled.
- Multi-instance operation (operable with several users).
- Multi-device operation (operable with several PROFIBUS controllers).
- Event handling on different priority levels according to user needs.
- Maximum user data length: 244 bytes for inputs, 244 bytes for outputs.
- Data consistency over the maximum user data length of 244 bytes.
- Baudrates from 9.6 kBaud up to 12 MBaud.

Environment:

- Independence of the operating system used
- Portability for processor families 8031, 80C16x, i80x86, Pentium (or compatible), and others
- Integration of the firmware in compact/modular slave applications

The figure below shows the placement of the V1SL firmware components (gray-shaded) in a slave module.

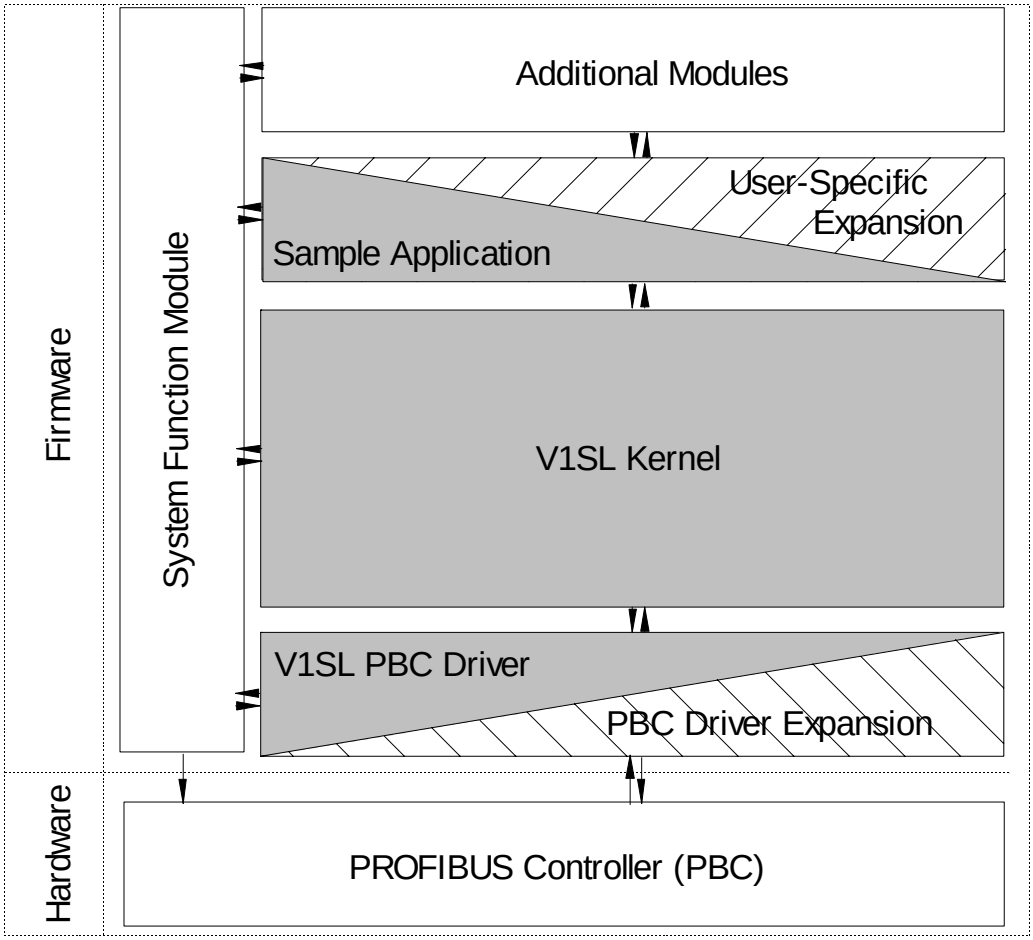


Figure 2: Placement of the V1SL in a Slave Module

Thus, the implementation effort by the user consists in the setting up his priority level system and his application that controls the DPV1 slave.

In addition, a system adaptation is required that initializes the V1SL, and makes basic resources available.

Below, the individual V1SL firmware components of the package are briefly described.

2.2 V1SL Kernel

This component implements the DPV1 slave functionality according to [2]. It represents the kernel of the firmware package, and is described in detail in the chapter 'V1SL Kernel'.

2.3 PBC Driver

The PBC driver firmware is the connecting link between the PROFIBUS controller (DPC31) and the V1SL kernel.
The details of the DPC31 driver are described in the chapter ‘PBC Driver’.

2.4 Application Examples

Regards to the different possibilities, getting started with the firmware package is made easier for the V1SL user with the included sample application (refer to ‘V1SL Getting Started’).

2.5 Delivery Package/Installation/Configuration/Generation

This point provides notes regarding the installation and configuration of the components that the user has determined are suitable for him. The generation notes that follow comprise the last step prior to generating the firmware. These items are contained in the chapter ‘Application’ .

3 Structure

V1SL is functionally divided into (refer to Figure 1):

- Component for implementing the system interface and general, internal functions.
- Kernel for implementing the DP standard/DPV1 functionality for communication of the slave with a Class 1 (parameterization) master. Below, this component is called **C0 Firmware**. This includes several components:

  - State machine for cyclic services of the slave **C0** (or MSCY1S)
  - Alarm state machine of the slave **AL** (or MSAL1S)
  - State machine for acyclic services of the slave **C1** (or MSAC1S)

- Kernel for implementing the DPV1 functionality for communication of the slave with a Class 2 master. Below, this component is called **C2 Firmware** (or MSAC2S).
- Component for implementing the PBC driver system interface and general, internal PBC driver functions.
- Driver for implementing the interface to the PBC DPC31.

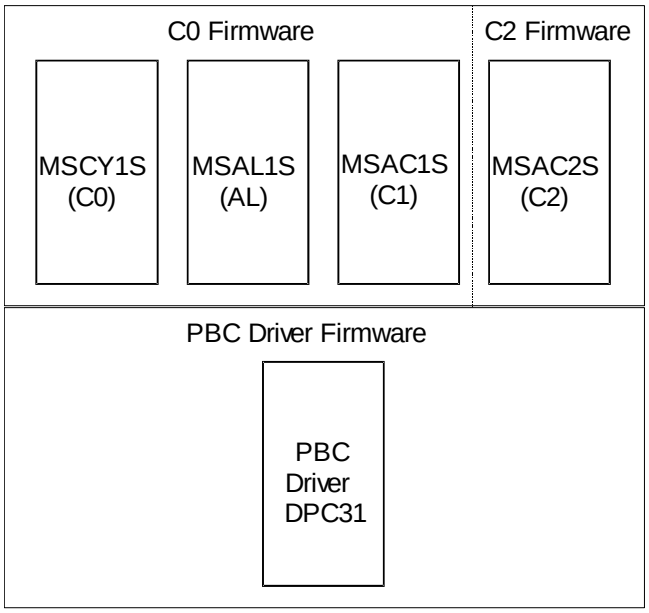


Figure 3: V1SL Components (without System Interface)

4 Functions

4.1 General

The DPV1 slave has interfaces to the system environment and to the user (application), as shown in Figure 2. Two cases are to be distinguished here:

- There are interface functions in the DPV1 slave that are called by the user or the system. These functions are called input functions of the DPV1 slave.
- Also, the DPV1 slave has to call user or system functions. From the view of the DPV1 slave, these are output functions.

Since the DPV1 slave is to be used in different applications with individual system environments, these output functions have to be designed in a way so they can be adapted. Therefore the DPV1 slave output functions are laid out as

- Output macros that have to be replaced in corresponding configuration files by the concrete functions of the user or the system environment
- Call back functions (CBF) which the user has to make available in the form of function pointers when opening a communication channel.

Figure 2 below shows the V1SL interfaces to the adjacent components. Input functions are shown as a number within a circle and output macros as a number within a square. The assignment of the interfaces represented by numbers to function and macro names is provided in the tables below.

It is to be pointed out that an output macro and an input function often form a functional pair (e.g. in the sense of request/acknowledgement). In this case, the relationship is indicated in the tables below by being shown in the same row. This assignment is not always unique; that is, an output macro may be assigned to different input functions, and vice versa.

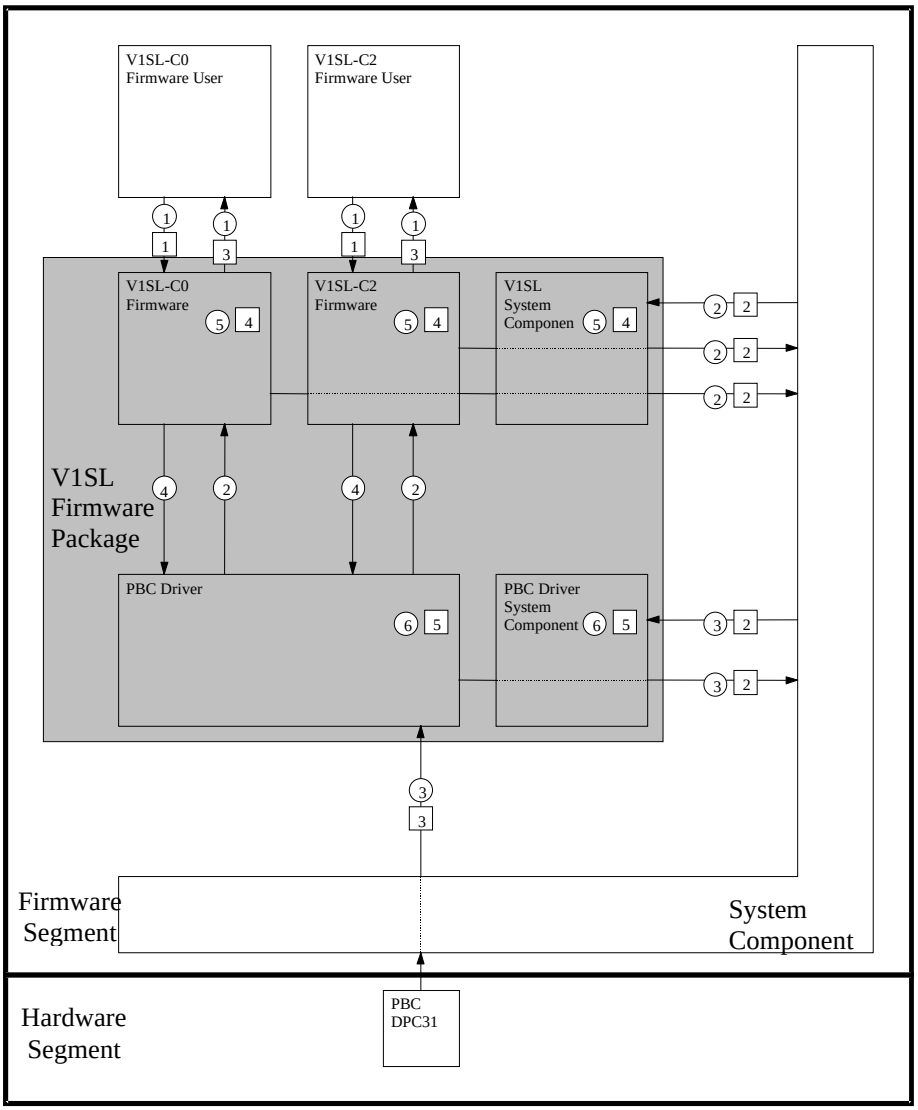


Figure 4: Interface Overview of V1SL

4.2 Identifiers

The identifier prefixes shown in Table 1 were selected on the basis of the name assignment of the V1SL firmware components.

Lower case letters are used for
- Input functions
- Variable identifiers

Upper case letters are used for
- Output macros
- Structure identifiers
- Value identifiers
- Attribute identifiers

Identifier Prefix	Concerns
v1sl_... / V1SL_...	All DPV1 slave firmware identifiers, particularly those that refer to general functionalities, and are not assigned to any of the individual components.
v1sl_c0_... / V1SL_C0_...	Identifiers for input functions/output macros that are assigned to the state machine for cyclic services (C0, MSCY1S).
v1sl_al_... / V1SL_AL_...	Identifiers for input functions/output macros that are assigned to the alarm state machine (AL, MSAL1S).
v1sl_c1_... / V1SL_C1_...	Identifiers for input functions/output macros that are assigned to the state machine for acyclic C1 services (C1, MSAC1S).
v1sl_c2_... / V1SL_C2_...	Identifiers for input functions/output macros that are assigned to the state machine for acyclic C2 services (C2, MSAC2S).
pbc_... / PBC_...	All PBC driver firmware identifiers that refer to general functionalities, and are not assigned to any of the specific PBC driver single components.
pbc_dpc31_...	Identifier for input functions that are assigned to the DPC31 PBC driver.

Table 1: Identifier Prefixes for V1SL

4.3 V1SL Interface to the System

4.3.1 Generally Utilized System Interface Functions/Macros

The numbers used in the table below at the beginning of the columns correspond to those in Figure 2. Numbers within circles represent input functions, which are shown here as (x). The output macros are shown as numbers within squares, which are shown here in the form [x].

For further description, refer to Section 9.1.

Input Function	Output Macro	Meaning
(2) v1sl_init		Initialization of the V1SL firmware package.
(2) v1sl_get_version		Read the V1SL firmware package version, and the installed components.
	[2] V1SL_ENTER	Disable V1SL calls.
	[2] V1SL_EXIT	Enable V1SL calls.
	[2] V1SL_FATAL_ERROR	Indicate a fatal error by V1SL.

Table 2: General System Interface Functions/Macros

4.3.2 System Interface Functions/Macros utilized by the C0/C2 Firmware

The numbers used in the table below at the beginning of the columns correspond to those in Figure 2. Numbers within circles represent input functions which are shown here as (x). The output macros are shown as numbers within squares, which are rendered here in the form [x].

For further description, refer to Section 9.2.

Input Function	Output Macro	Meaning
(4) v1sl_c0c1_perform_services		Event processing C0 firmware
(4) v1sl_c2_perform_services		Event processing C2 firmware
	[2] V1SL_C0C2_GET_PATH_INFO	Get the communication path information
	[2] V1SL_C0C2_RELEASE_PATH_INFO	Release the communication path information

Table 3: C0/C2 Firmware System Interface Functions/Macros

4.3.3 System Interface Functions/Macros utilized by the PBC Driver

The numbers used in the table below at the beginning of the columns correspond to those in Figure 2. Numbers within circles represent input functions which are shown here as (x). The output macros are shown as numbers within squares, which are rendered here in the form [x].

For further description, refer to Chapter ´PBC Driver´

Input Function	Output Macro	Meaning
(3) pbc_open_device		Make PBC known to the driver firmware
(3) pbc_close_device		Close PBC regarding utilization by V1SL as slave
(3) pbc_get_wd_state		Read watchdog status of a PBC
(3) pbc_get_baudrate		Read baudrate of a PBC
(3) pbc_trigger_user_wd		Trigger user watchdog
(3) pbc_dpc31_int_handler		DPC31 PBC interrupt handler
	[4] PBC_C0C1_EVENT_INDICATION	Event indication to C0 firmware
	[4] PBC_C2_EVENT_INDICATION	Event indication to C2 firmware

Table 4: PBC Driver System Interface Functions/Macros

4.4 V1SL Standard Interface to the User

4.4.1 C0 Firmware User Interface Functions/Macros

The numbers used in the table below at the beginning of the columns correspond to those in Figure 2. Numbers within circles represent input functions which are shown here as (x). The output macros are shown as numbers within squares, which are rendered here in the form [x].

For further description, refer to Section ´V1SL-Interface to the User´.

Input Function	Output Macro	Meaning
		User Interface C0
(1) v1sl_c0_open_channel	[1] V1SL_C0_OPEN_CHANNEL_DONE	Open a communication channel
(1) v1sl_c0_close_channel	[1] V1SL_C0_CLOSE_CHANNEL_DONE	Close a communication channel
(1) v1sl_c0_add		Allocation of the memory resources for the slave
(1) v1sl_c0_withdraw	[1] V1SL_C0_WITHDRAW_DONE	Enable of the slave memory resources
(1) v1sl_c0_control		Control the slave
	[1] V1SL_C0_DP_WD_TIMEOUT	Indicate expiration of DP watchdog timer
	[1] V1SL_C0_WD_STATE_REPORT	Indicate DP watchdog state change
	[1] V1SL_C0_DP_STATE_REPORT	Indicate DP state change
	[1] V1SL_C0_LED_STATE_REPORT	Indicate bus error LED state
	[1] V1SL_C0_DATA_EXCHANGE_ACTIVE	Signal slave's data traffic on the bus
	[1] V1SL_C0_NEW_SSA	Indicate slave address data
	[1] V1SL_C0_NEW_PRM	Indicate parameterization data
	[1] V1SL_C0_NEW_CFG	Indicate configuration data sent by the master
	[1] V1SL_C0_CLEAR	Indicate CLEAR/UNCLEAR
	[1] V1SL_C0_SYNC	Indicate SYNC/UNSYNC
	[1] V1SL_C0_FREEZE	Indicate FREEZE/UNFREEZE
(1) v1sl_c0_get_real_cfg_ptr		Get expected configuration data buffer
(1) v1sl_c0_real_cfg_update		Expected configuration update
	[1] V1SL_C0_REAL_CFG_BUFFER_ CHANGED	New expected configuration data buffer available to user
(1) v1sl_c0_calc_in_out_len		Calculate length of input and output data
(1) v1sl_c0_get_input_ptr		Get input data buffer
(1) v1sl_c0_input_update		Input data update
(1) v1sl_c0_get_output_info		Get output data buffer and buffer status information
(1) v1sl_c0_set_diag		Set diagnostic
	[1] V1SL_C0_DIAG_CHANGED	Return diagnostic buffer to user
	[1] V1SL_C0_DIAG_FETCHED	Indication 'diagnostic fetched from parameterization master'
		User interface AL
	[1] V1SL_AL_STATE_REPORT	Indicate activation/deactivation of the alarm state machine
(1) v1sl_al_set_alarm	[1] V1SL_AL_ALARM_ACK	Set/acknowledge alarm
(1) v1sl_al_withdraw_alarm	[1] V1SL_AL_ALARM_ACK	Withdraw/acknowledge alarm
		User interface C1
(1) v1sl_c1_read_ds_done	[1] V1SL_C1_READ_DS	Read data set via C1 firmware
(1) v1sl_c1_write_ds_done	[1] V1SL_C1_WRITE_DS	Read data set via C1 firmware

Table 5: C0 Firmware User Interface Functions/Macros

4.4.2 C2 Firmware User Interface Functions/Macros

The numbers used in the table below at the beginning of the columns correspond to those in Figure 2. Numbers within circles represent input functions, which are shown here as (x). The output macros are shown as numbers within squares, which are rendered here in the form [x].

For further description, refer to Section ´V1SL Standard Interface to the User´.

Input Function	Output Macro	Meaning
		User- Interface C2
(1) v1sl_c2_open_channel	[1] V1SL_C2_OPEN_CHANNEL_DONE	Open a communication channel
(1) v1sl_c2_close_channel	[1] V1SL_C2_CLOSE_CHANNEL_DONE	Close a communication channel
(1) v1sl_c2_initiate_done	[1] V1SL_C2_INITIATE	Indication/Acknowledge Initiate PDU
	[1] V1SL_C2_ABORT	Indication/Acknowledge Abort PDU
(1) v1sl_c2_user_abort	[1] V1SL_C2_USER_ABORT_DONE	User cancels connection
(1) v1sl_c2_data_transport_done	[1] V1SL_C2_DATA_TRANSPORT	Indication/Acknowledge Transport PDU
(1) v1sl_c2_read_ds_done	[1] V1SL_C2_READ_DS	Read data set via C2 firmware
(1) v1sl_c2_write_ds_done	[1] V1SL_C2_WRITE_DS	Write data set via C2 firmware

Table 6: C2 Firmware User Interface Functions/Macros

5 Operating Sequences

5.1 Initialization and Termination

The DPV1 slave firmware is initialized in several steps, as Table 7 shows. After the basic initialization of the entire package with the function *v1sl_init()*, the PROFIBUS controller is initialized with *pbc_open_device()*. Then, the communication channels for the C0 and/or C2 firmware are opened. The functions *v1sl_c0_open_channel()* and *v1sl_C2_open_channel()* are provided for this, or their equivalents when using the request block interface.
Applications that require termination of the DPV1 slave firmware can perform this also in several steps. For this, the V1SL provides the functions *v1sl_c0_close_channel()* and *v1sl_c2_close_channel().* The counterpart to the function *pbc_open_device()* is *pbc_close_device()*.

Based on the relationships described above between input functions and output macros, the user (application) has to execute the following sequence of function calls/ acknowledgements with the DPV1 slave (the numbers preceding the functions/macros correspond to the interface for the V1SL firmware package; refer to Figure 2).

		Power-Up/Shut Down PBC/C0/C2
User	**DPV1-Slave**	**System Calls**
	(2) v1sl_init() →	
	(3) pbc_open_device() →	
	(1) v1sl_c0_open_channel() →	V1SL_C0C2_GET_PATH_INFO() V1SL_PBC_GET_PATH_INFO()
	← [1] V1SL_C0_OPEN_CHANNEL_DONE()	
	(1) v1sl_c0_add() →	
	← [1] V1SL_C0_WD_STATE_REPORT()	
	← [1] V1SL_C0_DP_STATE_REPORT()	
	← [1] V1SL_C0_LED_STATE_REPORT()	
	← [1] V1SL_C0_REAL_CFG_BUFFER_CHANGED()	
	(1) v1sl_c0_get_real_cfg_ptr() →	
	(1) v1sl_c0_real_cfg_update() →	
	... (1) v1sl_c2_open_channel() →	V1SL_C0C2_GET_PATH_INFO() V1SL_PBC_GET_PATH_INFO()
	← [1] V1SL_C2_OPEN_CHANNEL_DONE()	
	... **Productive operation between User and DPV1-Slave (see below)** ...	
	(1) v1sl_c0_withdraw() →	
	← [1] V1SL_C0_WITHDRAW_DONE()	
	(1) v1sl_c0_close_channel() →	V1SL_PBC_RELEASE_PATH_INFO() V1SL_C0C2_RELEASE_PATH_INFO()
	← [1] V1SL_C0_CLOSE_CHANNEL_DONE()	
	... (1) v1sl_c2_close_channel() →	V1SL_PBC_RELEASE_PATH_INFO() V1SL_C0C2_RELEASE_PATH_INFO()
	← [1] V1SL_C2_CLOSE_CHANNEL_DONE()	
	(3) pbc_close_device() →	

Table 7:Initialization/Termination of V1SL

This method ensures the following features of the slave package:

- Multi instance/multi device operation is supported (refer to the chapters 'Multi Instance Operation', and 'Multi Device Operation')
- Certain details (e.g. the active SAPs of the C2 firmware that are used) can be specified at program runtime.
- If possible, resources are assigned only when they are actually needed.
- If needed, the user can initiate the 'return path' by closing a communication channel (releasing assigned resources).

The information needed by V1SL when setting up a communication channel is expected in the form of a data structure called detail block.
The system environment of the V1SL is informed of this structure. The system environment has to set up a data with this structure, and enter the details of a communication channel.

The detail block is not directly visible to the firmware. In the context of processing the functions *v1sl_c0_open_channel()* or *v1sl_c2_open_channel()*, the output macros *V1SL_C0C2_GET_PATH_INFO()* and *V1SL_PBC_GET_PATH_INFO()* are called (refer to Figure 3). The system environment has to transfer the detail block in form of a detail pointer. An additional parameter of the output macros is a system pointer for identifying the layer below, which the V1SL firmware package itself does not need. In the case of some system output macros, V1SL provides this pointer to the system environment.

For general use (multi instance/multi device operation), it is necessary to differentiate between instances (communication channels) and their connection over all communication layers (layer stack). For that reason, the functions *v1sl_c0_open_channel()* or *v1sl_c2_open_channel()* have a parameter *sys_path*. This parameter specifies a complete communication path through the layer stack; that is, the 'interconnection' of the instances of stacked firmware layers. By selecting a type for *sys_path*, the system environment itself can specify how it wants to differentiate between the different communication paths (e.g. *Unsigned8*).

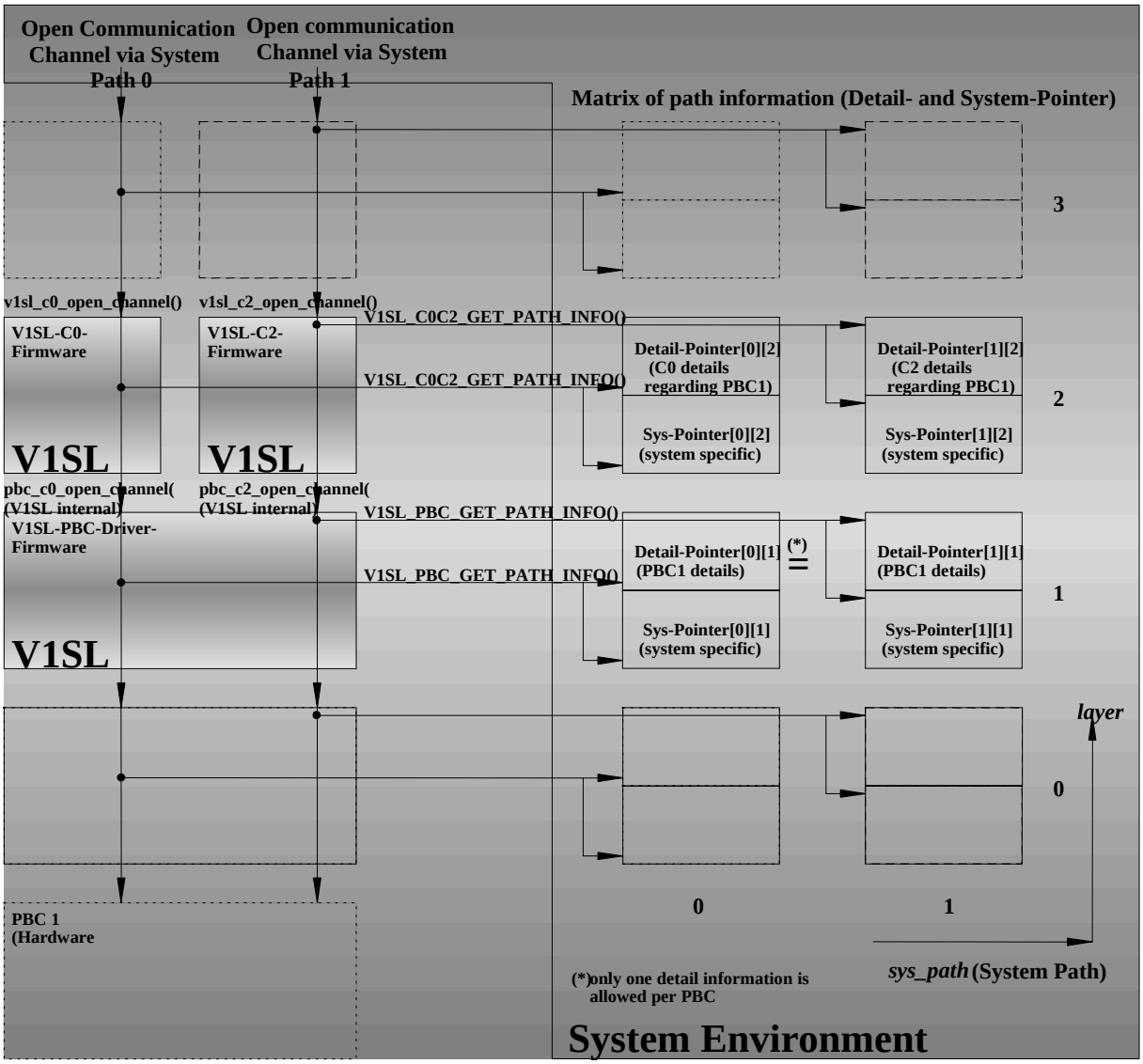
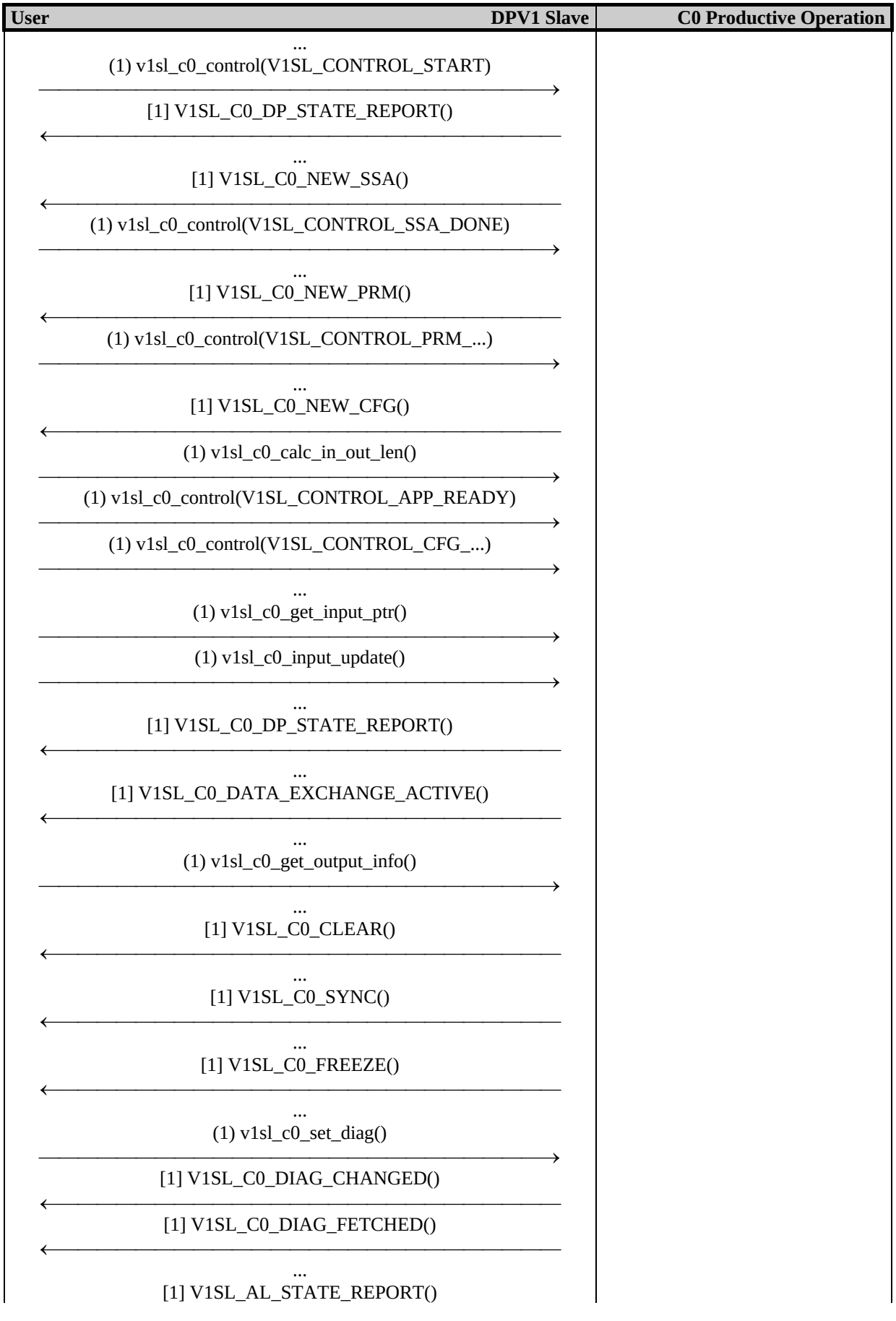


Figure 5: System Paths (*sys_path)* through Layer Stack

5.2 Operation of the C0 Firmware

Based on the relationships described above between input functions and output macros, the user (application) can execute the following sequence of function calls/ acknowledgements with the DPV1 slave (the numbers preceding the functions/macros correspond to the interface for the V1SL firmware package; refer to Figure 4).



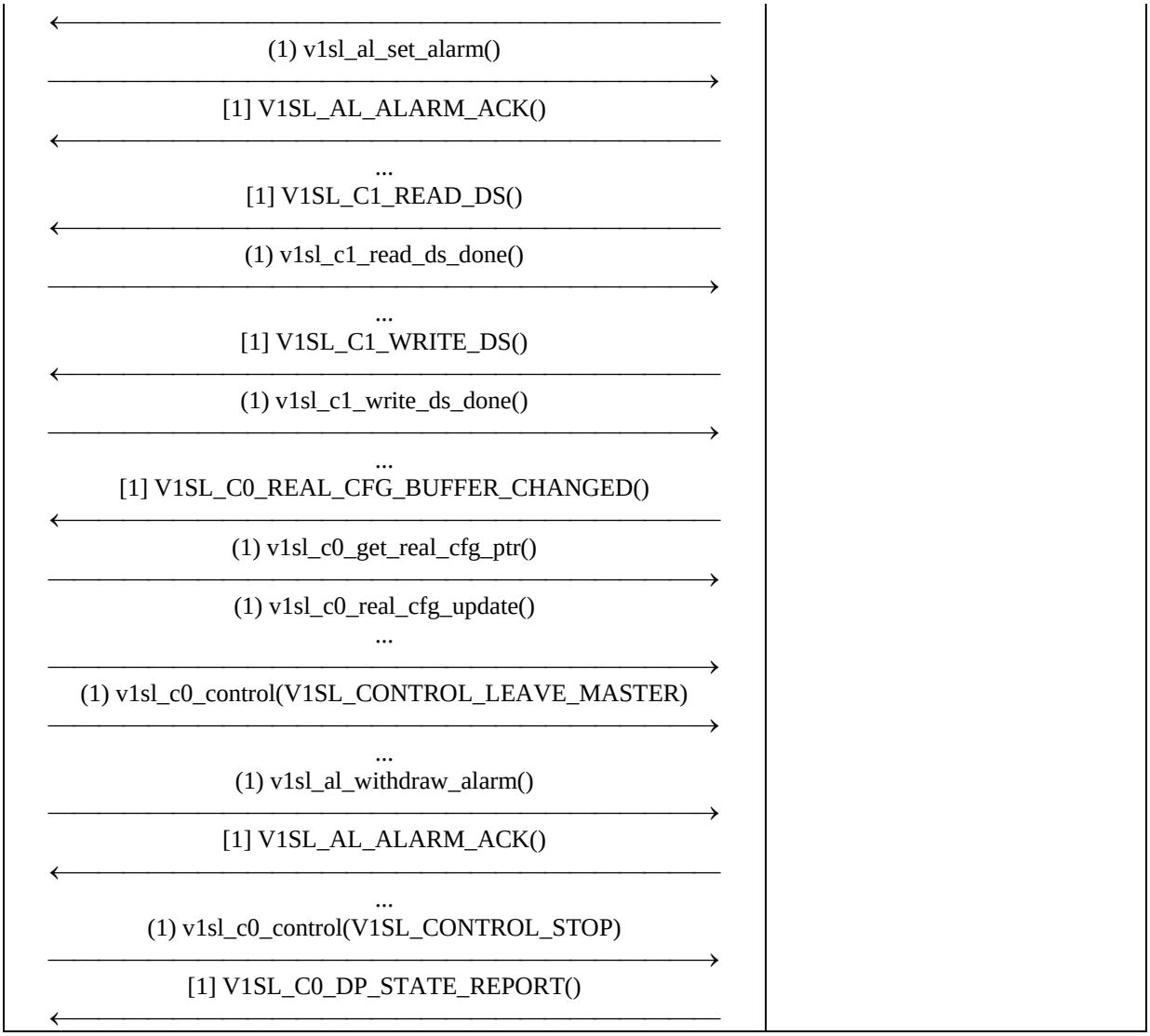


Table 8: Operation of the C0 Firmware

5.3 Operation of the C2 Firmware

Based on the relationships described above between input functions and output macros, the user (application) can execute the following sequence of function calls/ acknowledgements with the DPV1 slave (the numbers preceding the functions/macros correspond to the interface for the V1SL firmware package; refer to Figure 4).

User	DPV1 Slave	C2 Productive Operation
[1] V1SL_C2_INITIATE() ←		
(1) v1sl_c2_initiate_done() →		
...		
[1] V1SL_C2_DATA_TRANSPORT() ←		
(1) v1sl_c2_data_transport_done() →		
...		
[1] V1SL_C2_READ_DS() ←		
(1) v1sl_c2_read_ds_done() →		
...		
[1] V1SL_C2_WRITE_DS() ←		
(1) v1sl_c2_write_ds_done() →		
...		
[1] V1SL_C2_ABORT() ←		
OR		
(1) v1sl_c2_user_abort() →		
[1] V1SL_C2_USER_ABORT_DONE() ←		

Table 9: Operation of the C2 Firmware

6 Bases of C0 Firmware Utilization

6.1 Slave State Machine

The state machine that handles the C0 firmware visible to the user is shown in Figure 6.
To signal DP state transitions to the user, the C0 firmware provides the output macro *V1SL_C0_DP_STATE_REPORT()*. The user can influence the DP state by calling the input function *v1sl_c0_control()* in different situations.

Note: Not only the user, but also the bus can influence the DP state. This is not shown in Figure 6.

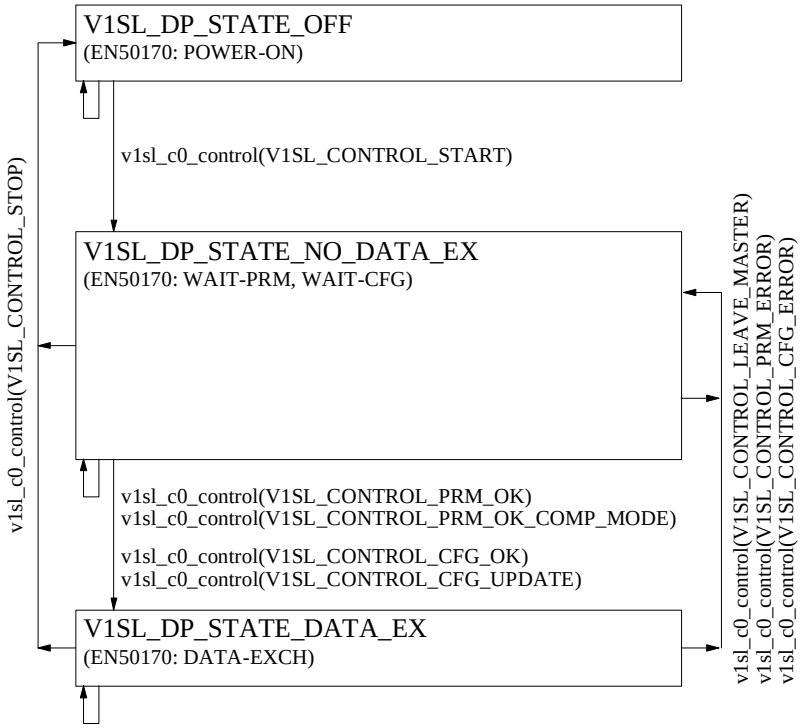


Figure 6: Slave State Transitions, and Influencing User Actions

6.2 Slave Parameterization

6.2.1 General

The manufacturer specifies individual slave parameters. Therefore a device description file (GSD) has to be created for every. The parameters specified in this respect reach the slave via the parameterization message of the parameterization master. Here, it is to be ensured that after the first 7 bytes of the parameterization message, the 3 DPV1 status bytes have to be provided in addition. In order to ensure startup also with DP standard masters that exclusively supply 7 bytes parameterization data according to EN50170, the DPV1 status bytes are accepted within the DPV1 slave with preset values which are not visible to the user. The DPV1 status bytes are protocol parameters whose structure is described below.

To indicate received parameterization data to the user, the firmware provides the output macro *V1SL_C0_NEW_PRM()*. The user can acknowledge the processing with *v1sl_c0_control(V1SL_CONTROL_PRM_...)*.

6.2.2 Structure of the Parameterization Data

The parameterization data consists of the following:

- DP standard parameters with a length of 7 bytes according to EN 50170.
- DPV1 status bytes with a length of 3 bytes according to extensions of EN 50170 which immediately follow the specified DP standard parameters.
- Optional user parameterization data that immediately follow the DPV1 status bytes. Its structure has not been specified within the scope of DPV1 standardization.

station_ state	wd_fact_1	wd_fact_2	mintsdr	pno_ ident_hi	pno_ ident_lo	group_ ident	DPV1_1 Bit 7..0	DPV1_2 Bit 7..0	DPV1_3 Bit 7..0

Table 10: Structure of the DPV1 Parameterization Message

Parameter	Value	Meaning
		DPV1 Status 1
DPV1_1.7	0_B 1_B	DP standard operation DPV1 operation
DPV1_1.6	0_B 1_B	The master does not operate the slave in the 'Failsafe Mode' The master operates the slave in the 'Failsafe Mode'
DPV1_1.5	0_B 1_B	Reserved
DPV1_1.4	0_B 1_B	Reserved
DPV1_1.3	0_B 1_B	Reserved
DPV1_1.2	0_B 1_B	DP Watchdog Basis: 10ms DP Watchdog Basis: 1ms
DPV1_1.1	0_B 1_B	Reserved
DPV1_1.0	0_B 1_B	Reserved
		DPV1 Status 2
DPV1_2.7	0_B 1_B	Pull and plug alarms disabled Pull and plug alarms enabled
DPV1_2.6	0_B 1_B	Process alarms disabled Process alarms enabled
DPV1_2.5	0_B 1_B	Diagnostic alarms disabled Diagnostic alarms enabled
DPV1_2.4	0_B 1_B	Manufacturer-specific alarms disabled Manufacturer-specific alarms enabled
DPV1_2.3	0_B 1_B	Status alarms disabled Status alarms enabled
DPV1_2.2	0_B 1_B	Update alarms disabled Update alarms enabled
DPV1_2.1	0_B 1_B	Reserved
DPV1_2.0	0_B	Configuration Mode: If there are differences between configuration sent by master and the expected configuration, the slave does not enter the data exchange mode (slave state

		V1SL_DP_STATE_DATA_EX)
	1_B	Even if there are differences between the configuration sent by master and the expected configuration, the slave enters the data exchange mode (slave state *V1SL_DP_STATE_DATA_EX*)
		DPV1 Status 3
DPV1_3.7	0_B 1_B	Reserved
DPV1_3.6	0_B 1_B	Reserved
DPV1_3.5	0_B 1_B	Reserved
DPV1_3.4	0_B 1_B	Reserved
DPV1_3.3	0_B 1_B	Reserved
DPV1_3.2 . . . DPV1_3.0	 000_B 001_B 010_B 011_B 100_B 101_B 110_B 111_B	Alarm Mode: Slave is operated in the alarm type mode; this means, only one alarm of the same type is to be active at the same time. Slave is operated in the alarm sequence mode; maximum number of alarms active at the same time: 2 Slave is operated in the sequence mode; maximum number of alarms active at the same time: 4 Slave is operated in the sequence mode; maximum number of alarms active at the same time: 8 Slave is operated in the sequence mode; maximum number of alarms active at the same time: 12 Slave is operated in the sequence mode; maximum number of alarms active at the same time: 16 Slave is operated in the sequence mode; maximum number of alarms active at the same time: 24 Slave is operated in the sequence mode; maximum number of alarms active at the same time: 32

Table 11: Structure of the DPV1 Status Byte in the Parameterization Message

6.2.3 Default Parameterization

Default parameterization makes it possible for old DP standard masters to exchange data with the DPV1 slave despite missing DPV1 status bytes. The parameterization message is 7 bytes long.

station_ state	wd_fact_1	wd_fact_2	mintsdr	pno_hi	pno_lo	group_ ident

In this case, the slave will use the following values for the missing DPV1 status bytes:

Parameter	Value	Meaning
		DPV1 Status 1
DPV1_1.7	0_B	DP standard operation
DPV1_1.6	0_B	The master does not operate the slave in the 'Failsafe Mode'
DPV1_1.5	0_B	reserved
DPV1_1.4	0_B	reserved
DPV1_1.3	0_B	reserved
DPV1_1.2	0_B	DP Watchdog Basis: 10ms
DPV1_1.1	0_B	Is not evaluated by the PBC driver firmware
DPV1_1.0	0_B	Is not evaluated by the PBC driver firmware
		DPV1 Status 2
DPV1_2.7	0_B	Pull and plug alarms disabled
DPV1_2.6	0_B	Process alarms disabled
DPV1_2.5	0_B	Diagnostic alarms disabled
DPV1_2.4	0_B	Manufacturer-specific alarms disabled
DPV1_2.3	0_B	Status alarms disabled
DPV1_2.2	0_B	Update alarms disabled
DPV1_2.1	0_B	reserved
DPV1_2.0	0_B	If there is a difference between configuration sent by master and expected configuration, the slave does not enter the data exchange mode (slave state *V1SL_DP_STATE_DATA_EX*)
		DPV1 Status 3
DPV1_3.7	0	reserved
DPV1_3.6	0	reserved
DPV1_3.5	0	reserved
DPV1_3.4	0	reserved
DPV1_3.3	0	reserved
DPV1_3.2 ... DPV1_3.0	000_B	Slave is operated in the alarm type mode; that is, only one alarm of the same type is to be active at the same time.

Table 12: DPV1 Status generated within the DPV1 Slave in the case of default parameterization

6.3 Slave Configuration

6.3.1 General

The configuration possibilities is specified for each slave module in the device description file (GSD) The parameterization master sends the specified parameters to the slave within the configuration telegram. Also the slave provides the expected configuration to every master if requested.

The formats of the configuration data are specified in EN 50170, and are described briefly here.

Note: The manufacturer of a slave module should adhere to **one** format for the configuration data; that is, no mixed use of formats.

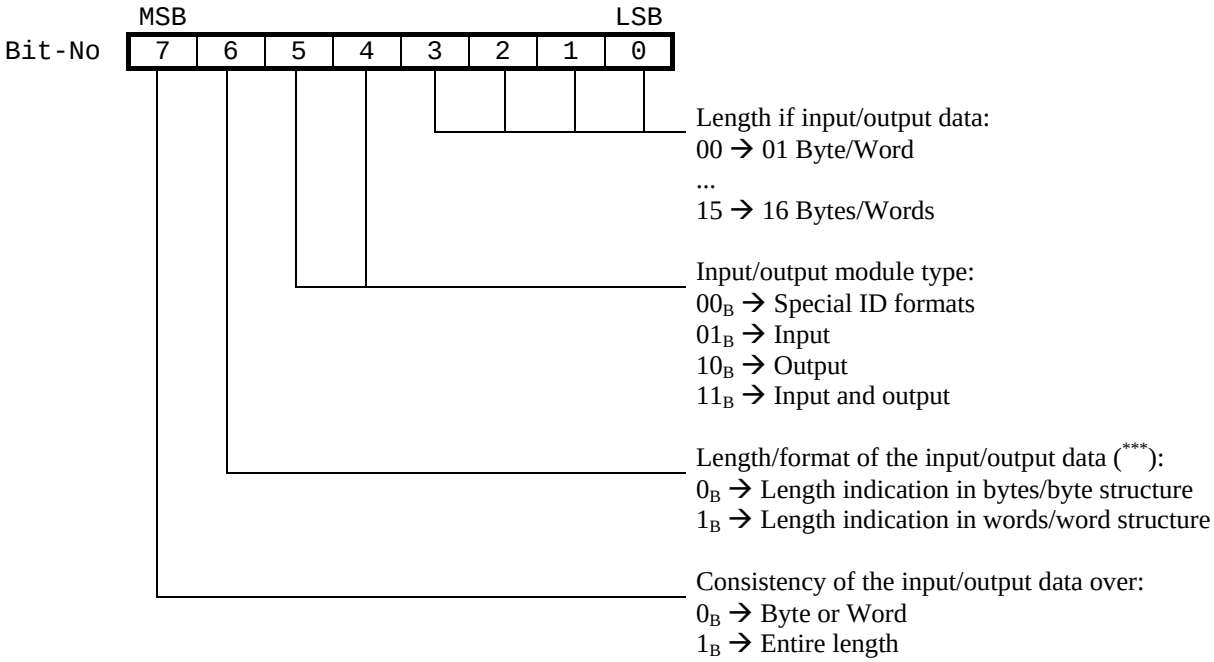
For the indication of received configuration data sent by the master to the user, the C0 firmware provides the output macro *V1SL_C0_NEW_CFG()*. The user can acknowledge the indication with *v1sl_c0_control(V1SL_CONTROL_CFG_...)*. In addition, the user can execute the input function *v1sl_c0_calc_in_out_len()* to

determine the input and output data length for user data requested by the parameterization master.

With the output macro *V1SL_C0_REAL_CFG_BUFFER_CHANGED()* and the input functions *v1sl_c0_get_real_cfg_ptr()* and *v1sl_c0_real_cfg_update()*, the user can provide the expected configuration data.

6.3.2 General ID Format

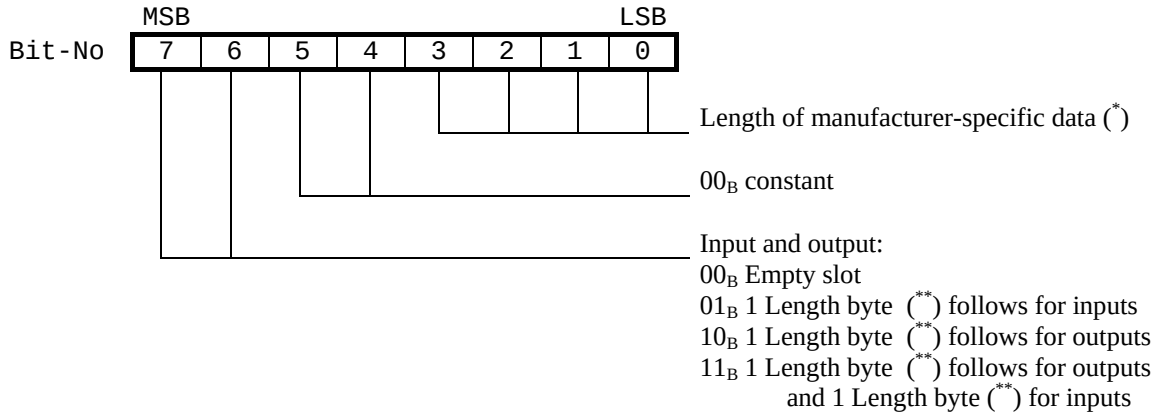
For modular slaves, an ID byte of this format should be used per module. This makes assigning module-specific diagnostics to the modules possible. The number (starting with the value 1) of the module *slot_number* in the diagnostic data corresponds to the number of the ID byte in the configuration data (starting with the value 1).
In the case of this format, the information regarding the length of the input- and output area is encoded in one byte.



(***) When words are transmitted, first the high byte and then the low byte is sent in the case of PROFIUS DP.

6.3.3 Special ID Format

Regarding this format, the information about the length of the input and output data area is encoded in one byte respectively. In addition, it is possible to embed manufacturer-specific data in the configuration per module.



(*) The length for manufacturer-specific data is to be interpreted as follows:

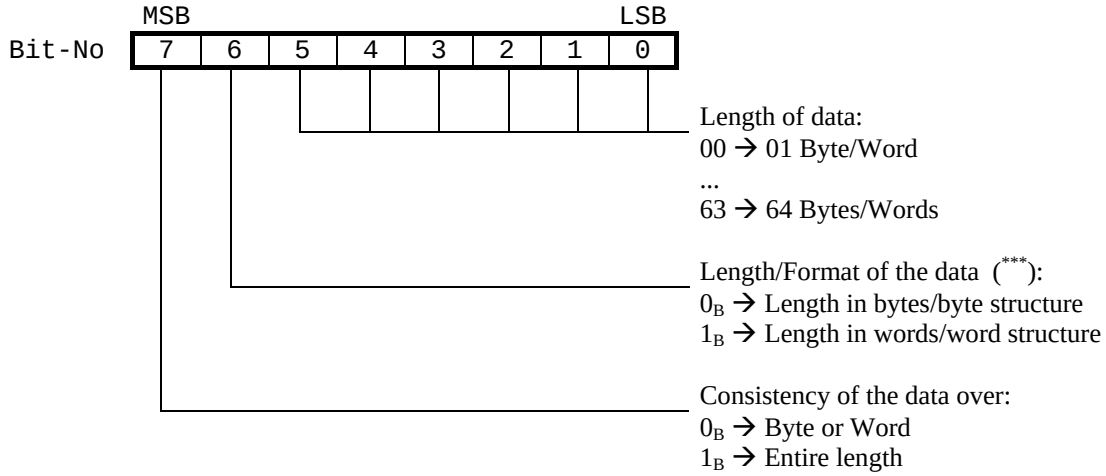
When configuration data sent by the master is indicated the following applies:

- 0: No manufacturer-specific data follows; also no manufacture-specific data in the expected configuration for this module
- 1..14: Manufacturer-specific data of the specified length follows; this data has to agree with the data in the expected configuration for this module.
- 15: No manufacturer-specific data follows; no check is made whether the expected configuration data includes manufacturer-specific data.

If this format is used for expected configuration data of the slave, the following applies:

- 0: No manufacturer-specific data follows.
- 1..14: manufacturer-specific data of the specified length follows.
- 15: not allowed.

(**) Below, the structure of the length bytes is shown:



(***) When words are transmitted, first the high byte and then the low byte is sent in the case of PROFIUS DP.

6.4 Slave Diagnostics and Slave Alarms

6.4.1 General

The implemented diagnostic concept supports the following types of diagnostics:

- Standard diagnostic
- Revision
- ID-related diagnostic
- Channel-related diagnostic
- Device-related diagnostic (primarily status messages)

The standard diagnostic which consists of 6 bytes is set up by the PBC driver firmware, or the PBC itself, and is not discussed in detail below. The user can only influence the value of the diagnostic bits *Ext-Diag*, *Ext-Diag-Overflow* and *Stat-Diag* within the standard diagnostic.
For user utilization of diagnostics, the C0 firmware provides the input function *v1sl_c0_set_diag()*, and the output macros *V1SL_C0_DIAG_CHANGED()* and *V1SL_C0_DIAG_FETCHED()*.

In addition, the alarm mechanism with extensions is used.

- Alarms are mapped to device-related diagnostics.
- Alarms can only be set by the user after activating the alarm state machine.
- Alarms are queued by the C0 firmware, and transmitted autonomously.
- Only one alarm is transmitted in a diagnostic telegram.
- Alarms that were set but not sent can be refetched by the user from the C0 firmware.
- After the alarm state machine was reset, alarms have to be fetched back by the user from the C0 firmware.

For user utilization of diagnostics, the C0 firmeware provides the input functions *v1sl_al_set_alarm()* and *v1sl_al_withdraw_alarm()* as well as the output macros *V1SL_AL_STATE_REPORT()* and *V1SL_AL_ALARM_ACK()*.

The general structure of the already mentioned diagnostic types including alarm is described in the sections below. The actual method used by the user for setting a diagnostic or an alarm by using specified data structures is described with the input functions *v1sl_c0_set_diag()* and *v1sl_al_set_alarm()*
According to the diagnostic message of the DPV1 slave consists of the following:

- **A static part** that the slave sends with each diagnostic requested by a master. The user can influence these diagnostic parts by calling the input function *v1sl_c0_set_diag()*:
  - ♦ No or one revision
  - ♦ No or ID-related diagnostic
  - ♦ No, one or several channel related diagnostics
  - ♦ No, one or several device-related diagnostics that are encoded as status message (not mandatory in DP standard operation).

- **A variable part** which the slave prepares only once in the diagnostic message. It contains an alarm that was set by the user. After the parameterization master has fetched it, this part can be deleted by the V1SL. However, this is the case only if the user in the meantime changes the static part of the diagnostic.

Figure 7 shows the implemented diagnostic and alarm concept.

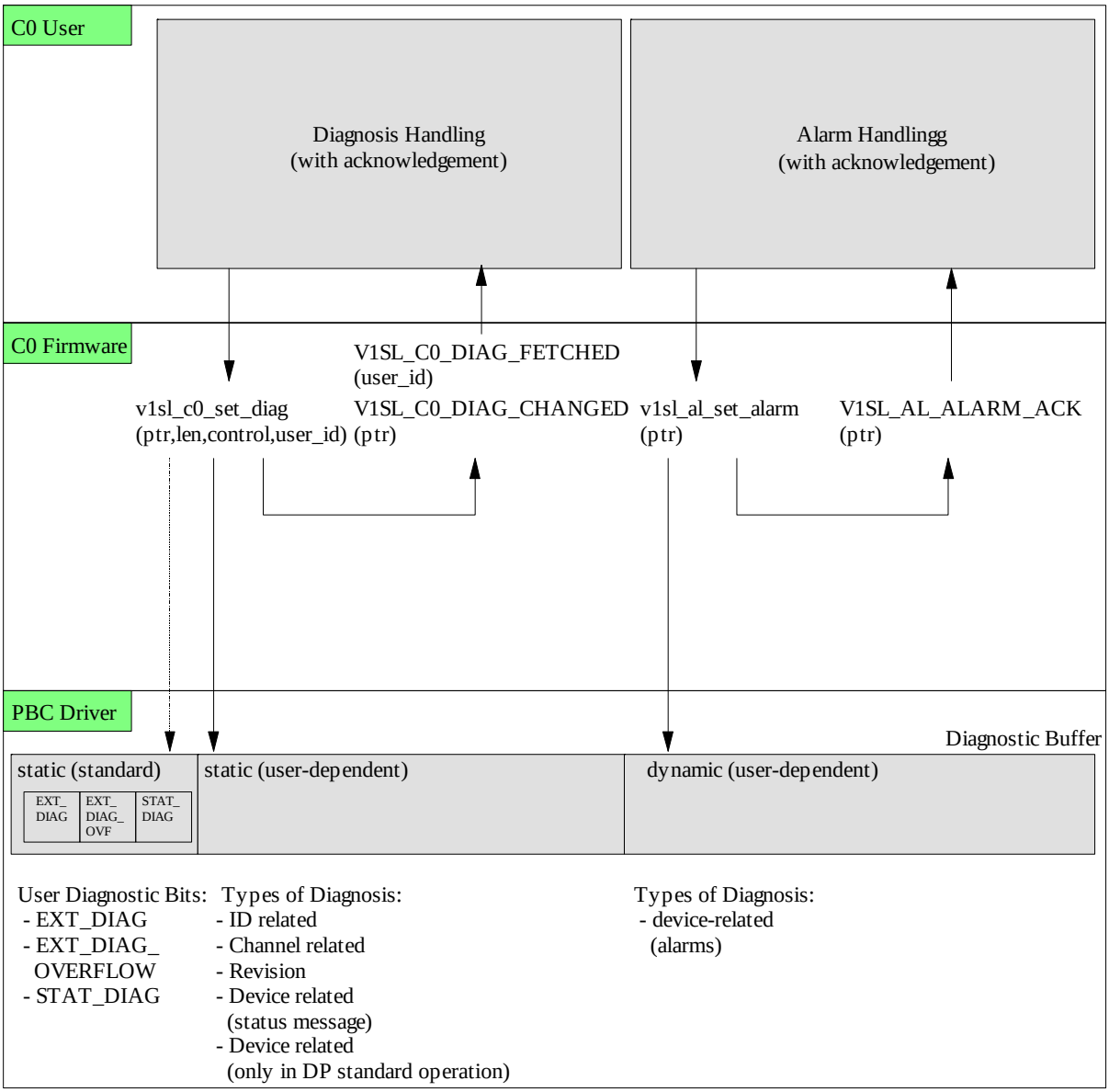


Figure 7: Diagnostic and Alarm Handling in V1SL

When initializing the DPV1 slave firmware, the user has to ensure that the diagnostic buffer is large enough for handling the diagnostics and alarms initialized by the user.

The figures below sketch the structure of the different diagnostic types that the user can generate. To complete the data, the 6 bytes standard diagnostic generated by the PBC driver firmware are also shown.

6.4.2 Structure of the Slave's Standard Diagnostic

Input function to be used: none (internal PBC driver processing!)
Operating mode: DP standard, DPV1

station_ status_1	station_ status_2	station_ status_3	master_ address	pno_ ident_hi	pno_ ident_lo

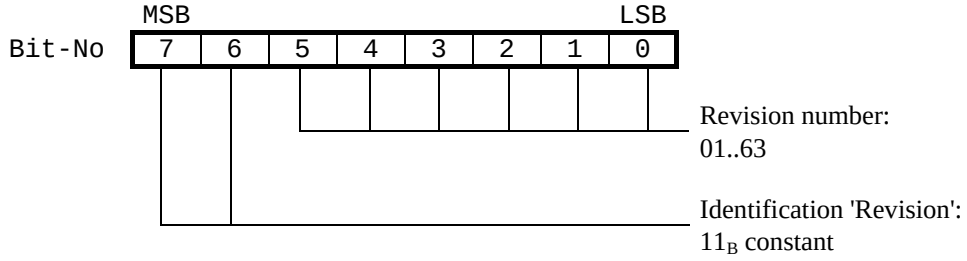
6.4.3 Structure of the Revision

Input Function to be used: *v1sl_c0_set_diag()*
Operating mode: DP standard, DPV1

The revision is used for checking whether the slave firmware/user application version is in accordance the device description file (GSD). The revision is 1 byte long.

sign_revision_number

The header byte *sign_revision_number* is to be encoded as follows:



Note: Only one revision is to be encoded in the diagnostic buffer.

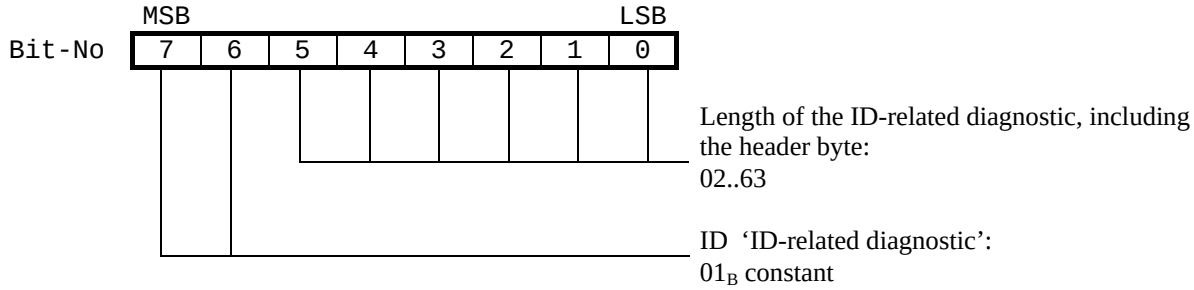
6.4.4 Structure of the ID-Related Diagnostic

Input Function to be used: *v1sl_c0_set_diag()*
Operating mode: DP-Norm, DPV1

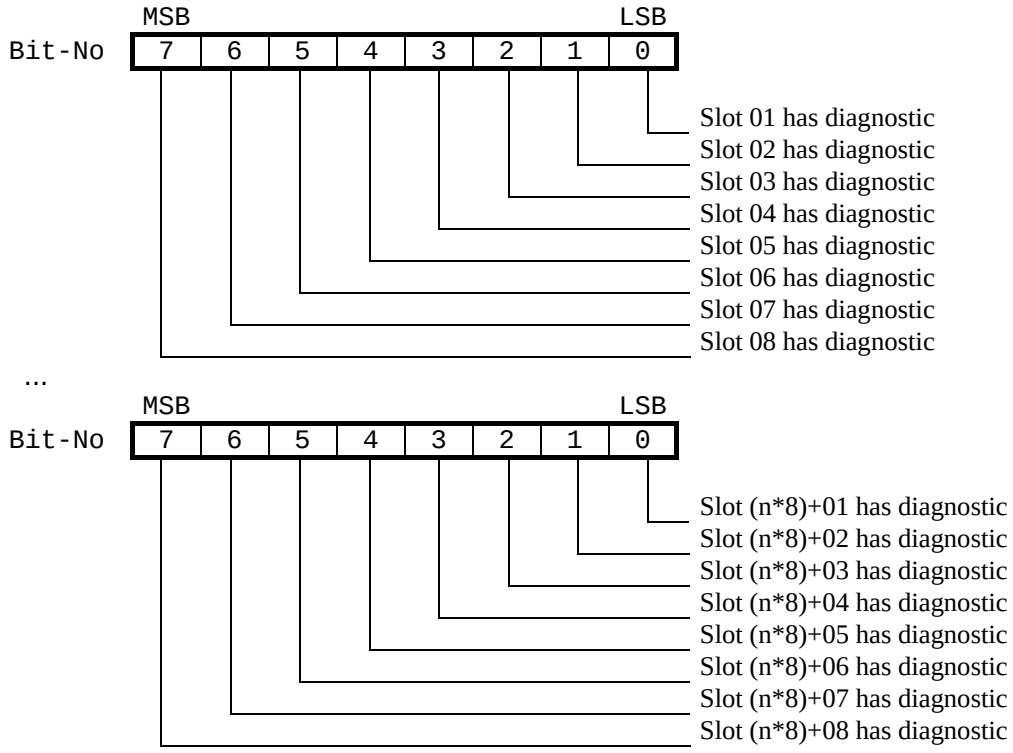
ID-related diagnostics are used to provide the master with an overview regarding which slots of a slave module are currently in a diagnostic mode. Each slot specified during configuration is assigned one bit. Bits not used are set to "0". A set bit means that in this input/output area a diagnostic is pending.

sign_len	slots[0] slot 01..08	slots[1] slot 09..16	Slots[2] slot 17..23	slots[3] slot 24..31	...	slots[n]

The header byte *sign_len* ist is to be encoded as follows:



In the data *slots[n]* , the information on the slots is to be encoded as follows:



Note: Only one ID-related diagnostic is to be encoded in the diagnostic buffer.

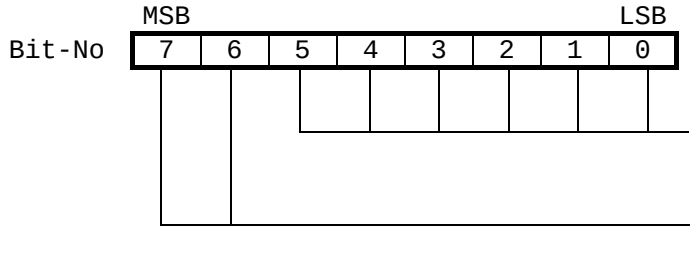
6.4.5 Structure of a Channel-Related Diagnostic

Input function to be used: *v1sl_c0_set_diag()*
Operating mode: DP-Norm, DPV1

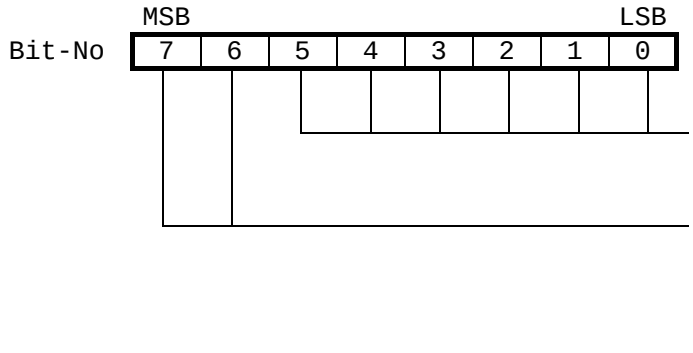
Channel-related diagnostics are used to specify the cause for the diagnostic for a certain slot or channel. For a channel, 3 diagnostic bytes are required.

sign_ident	number	code

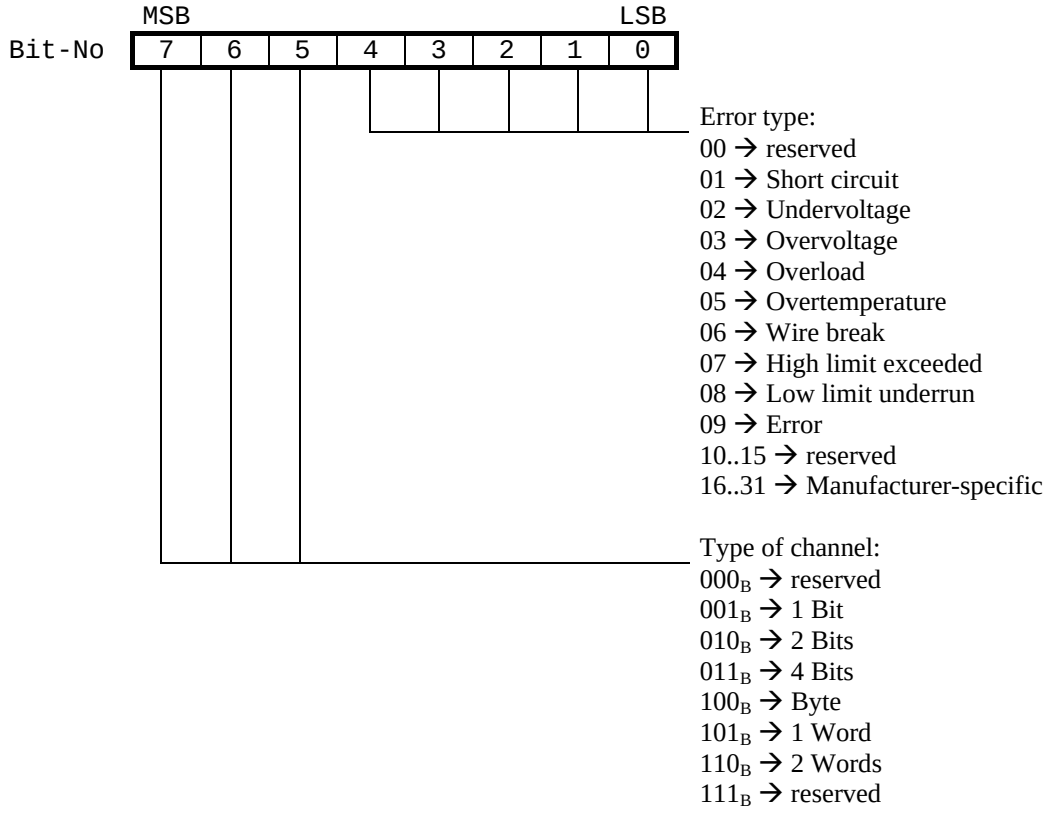
The header byte *sign_ident* is to be encoded as follows:



The byte *number* is to be encoded as follows:



The byte *code* is to be encoded as follows:



Note: Several channel-related diagnostics may be encoded in the diagnostic buffer.

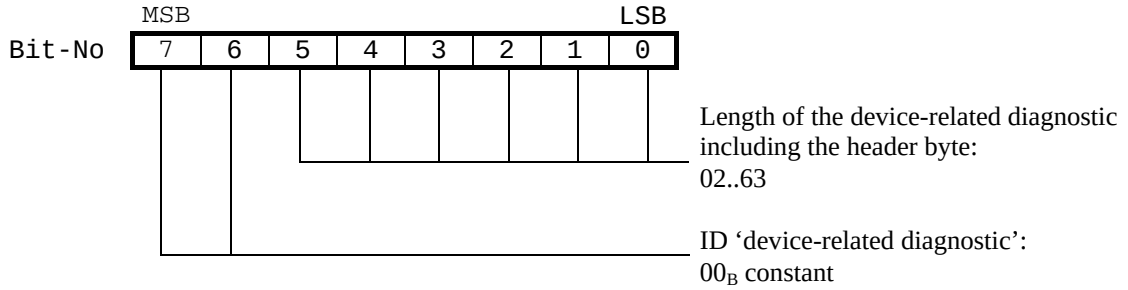
6.4.6 Structure of a Device-Related Diagnostic

Input function to be used: *v1sl_c0_set_diag()*
Operating mode: DP standard

The manufacturer specifies the structure of device-related diagnostics. As is the case with the ID-related and channel-related diagnostic, device-related diagnostics starts with a header byte:

sign_len	data_1	data_2	data_3	data_4	...	data_n

The header byte *sign_len* is to be encoded as follows:



Note: Only status PDUs and alarms are to be encoded as device-related diagnostics.

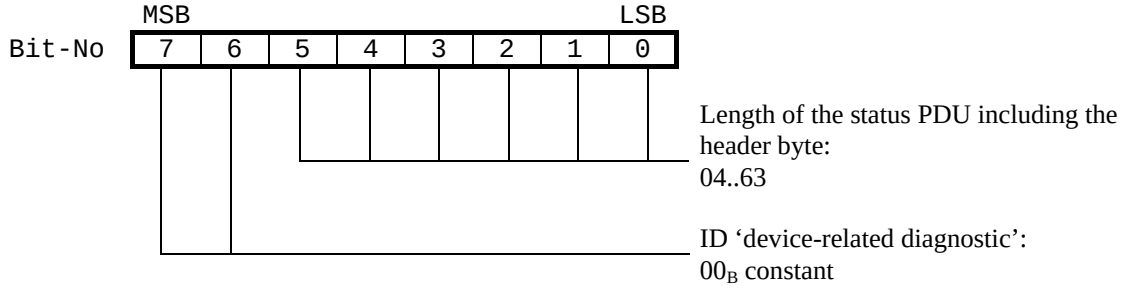
6.4.7 Structure of a Status PDU as Device-Related Diagnostic

Input function to be used: *v1sl_c0_set_diag()*
Operating mode: DPV1

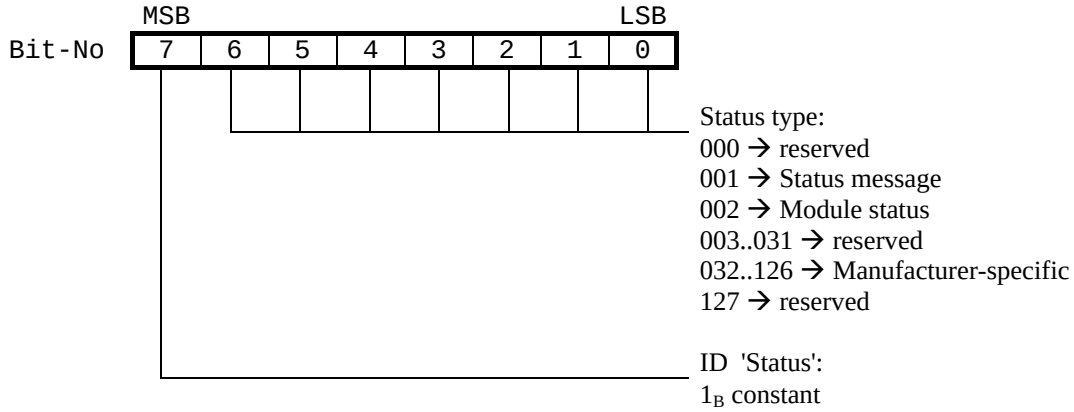
Status PDUs are mapped as device-related diagnostics; however, a certain structure is specified for the first 4 bytes. Status PDUs are recognized by the DPV1 master:

sign_len	status _type	slot _number	Specifier	user_ data 1	...	user_ data n

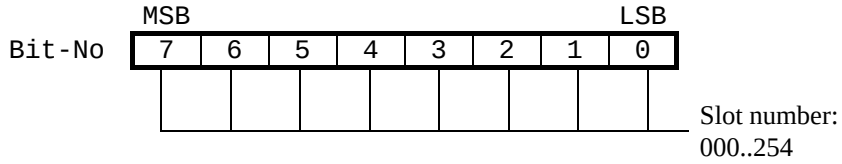
The header byte *sign_len* is to be encoded as follows:



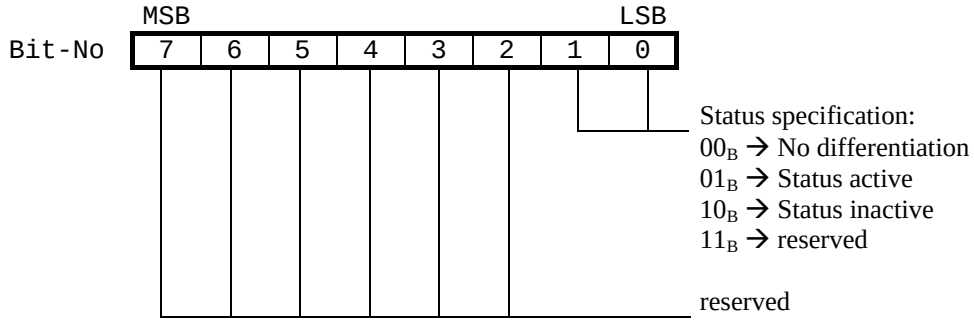
The header byte *status_type* is to be encoded as follows:



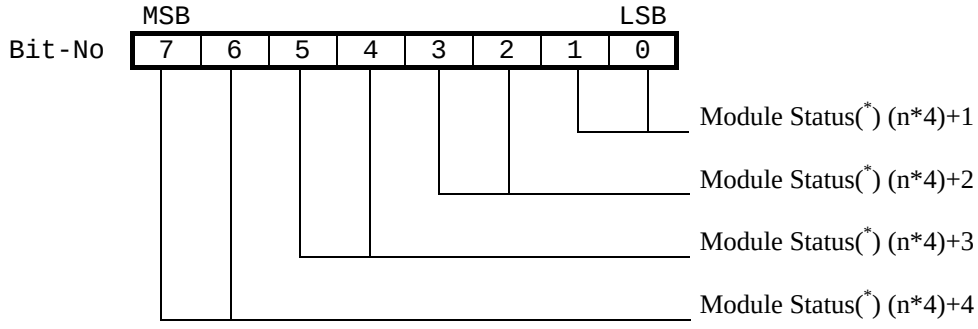
The header byte *slot_number* is to be encoded as follows:



The header byte *specifier* is to be encoded as follows:



If the value 'module status' is set in the byte *status_type*, the other data bytes are to be encoded as follows (n = 0..x):



(*)...The module status is to be encoded as follows:

Value	Meaning
00_B	Data valid
01_B	Data invalid, error in/at module
10_B	Data invalid, wrong module
11_B	Data invalid, no module

Otherwise, the additional bytes of the status PDU are to be defined manufactuer-specific.

Note: Several status PDUs may be encoded in the diagnostic buffer.

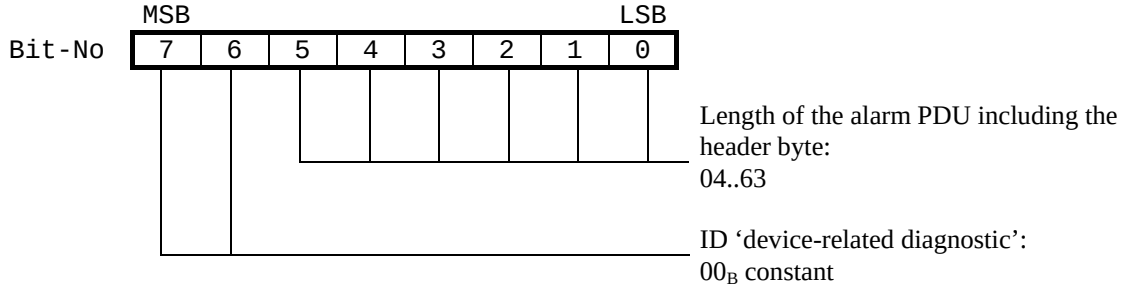
6.4.8 Structure of an Alarm PDU as Device-Related Diagnostic

Input function to be used: *v1sl_al_set_alarm()*
Operating mode: DPV1

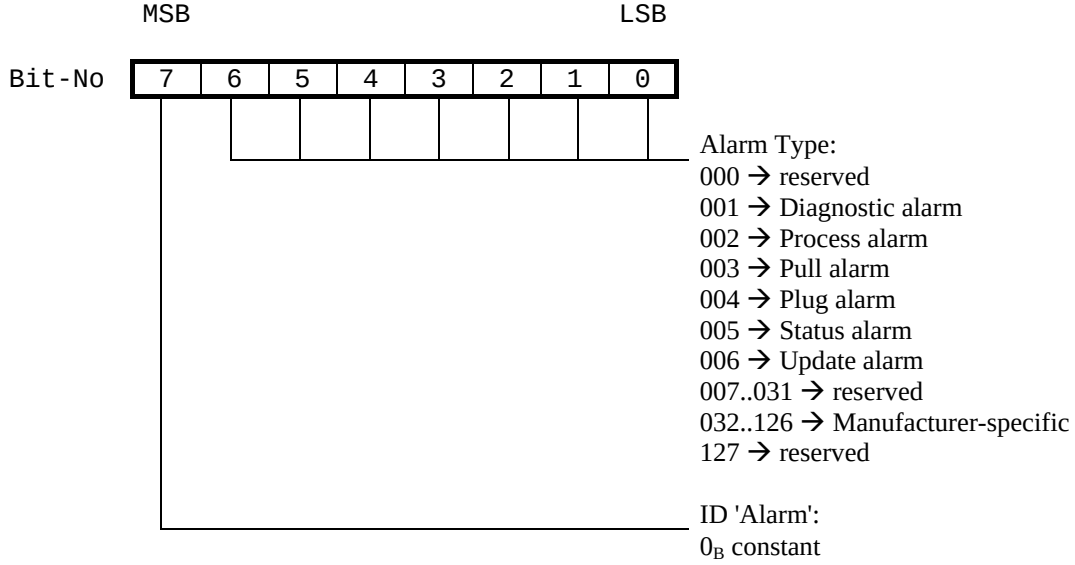
Alarm PDUs are mapped as device-related diagnostics, but for the first 4 bytes, a certain structure is specified. Alarm PDUs are recognized by the DPV1 master. The interpretation by DP standard masters depends on the device.

sign_len	alarm _type	slot _number	specifier	alarm_ data 1	...	alarm_ data n

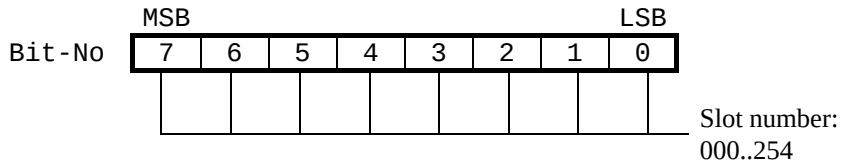
The header byte *sign_len* ist is to be encoded as follows:



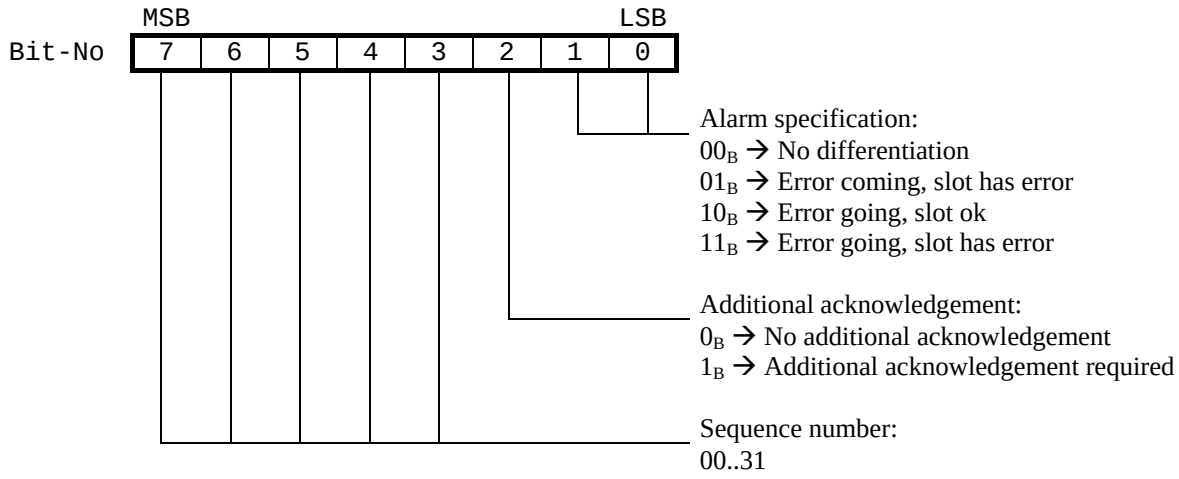
The header byte *alarm_type* is to be encoded as follows:



The header byte *slot_number* is to be encoded as follows:



The header byte *specifier* is to be encoded as follows:



The additional bytes of the alarm PDU are to be defined manufacturer-specific.

Note: Only one alarm PDU is to be encoded in the diagnostic buffer.

6.5 Slave Control by means of 'Application Ready'

The heading refers to a mechanism, redefined in the DPV1 draft for the user-controlled power-up of slave communication with the parameterization master.

Two states are differentiated:

- **Application Not Ready:** The slave enters this state after it has received configuration data from the parameterization master in the DP state *V1SL_DP_STATE_NO_DATA_EX*. If the slave enters the DP state *V1SL_DP_STATE_DATA_EX* after the data has been acknowledged positive by the user, there is still no user data exchange with the parameterization master. Instead, the slave requests static (continuous) reading of the slave diagnostic.
- **Application Ready:** After the user has triggered the cancellation of the state Appliation Not Ready, the slave deletes its static diagnostic request, and the parameterization master starts with the user data exchange.

Note: If the slave is not yet in the *V1SL_DP_STATE_DATA_EX* state when the user signals Application Ready, the static diagnostic bit does not become visible to the master. **Note:** If the slave receives configuration data sent by the master in the DP state *V1SL_DP_STATE_DATA_EX*, it is not possible to use the Application Ready mechanism.

To implement the Application Ready mechanism in the slave, the output macro *V1SL_C0_NEW_CFG()* and the input function *v1sl_c0_control(V1SL_CONTROL_APP_READY)* are used.

Extensive use of the Application Ready mechanism is recommended, but it is not mandatory. If the mechanism is not used, the user should adhere to the call sequence shown in Figure 8 below.

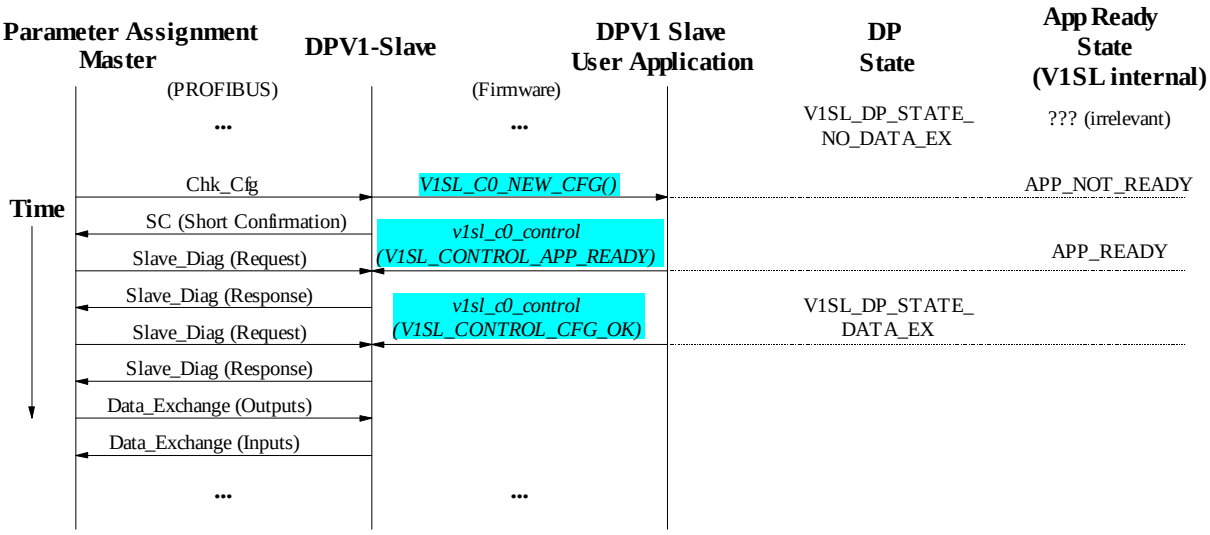


Figure 8: Without Application Ready Mechanism

The Application Ready mechanism makes it possible for the user to parameterize periphery modules (refer to Figure 7) prior to user data exchange (slave in DPV1 mode). This ensures that during the first user data transmission between master and slave, the value ranges of the transmitted user data are handled correctly.
The parameterization master carries out the transmission of the parameterization data for the periphery modules during the Application Not Ready phase by means of asynchronous (C1) services.

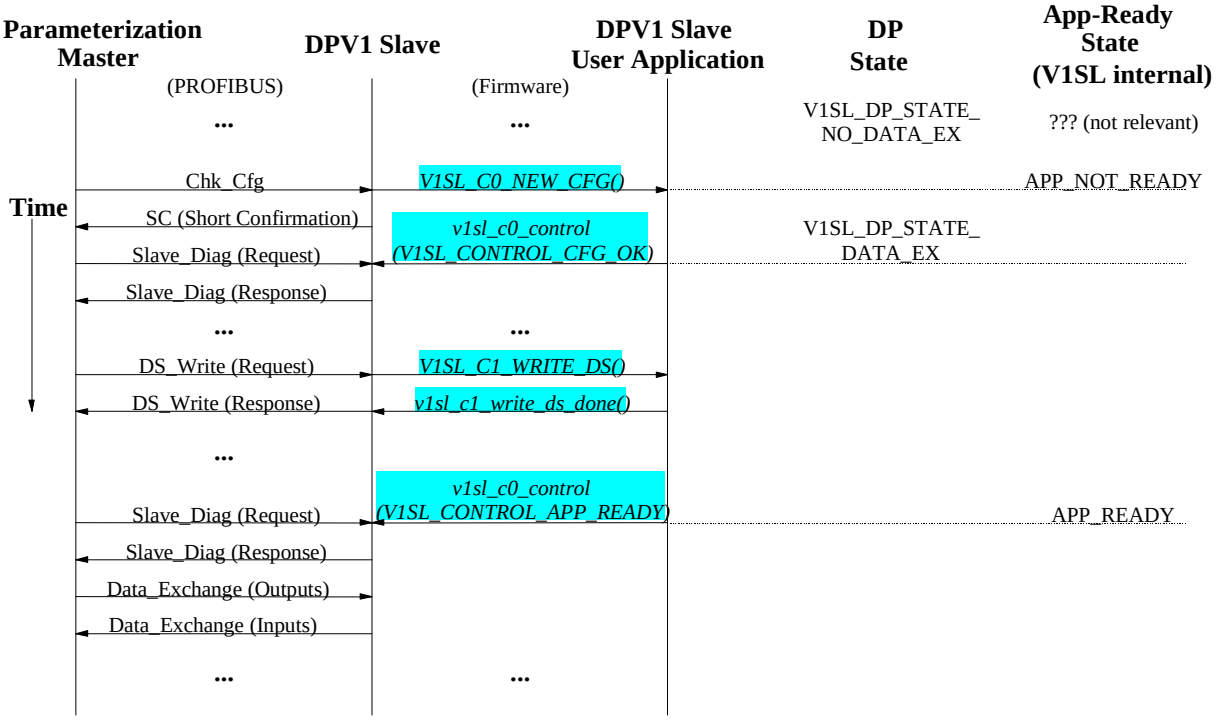


Figure 9: With Application Ready Mechanism

6.6 Handling Slave Output Data

Regarding the handling of output data, the DPV1 slave processes according to a strategy that makes it possible for the user to ignore cleared output data (values = 0).

Thus, the following substitute value strategy **can** be used:

- The user cyclically polls the slave for the presence of new output data. For this, the input function *v1sl_c0_get_output_info()* is used.
- If the returned state of the output data signals to the user that the parameterization master is in the state ‘clear‘ because of the ‘Global Control‘ command ‘Clear‘ (*V1SL_OUTPUT_STATE_GC_CLEAR*), or because it entered the 'Failsafe' mode (telegram with user data length 0, *V1SL_OUTPUT_STATE_CLEAR*), substitute values can be used:
  - ♦ Outputs are set to 0
  - ♦ Outputs will stay with the last value
  - ♦ Outputs are set to a parameterized substitute value
- If the parameterization master exits the ‘clear‘ state, this is signalled by the bit *V1SL_OUTPUT_STATE_GC_UNCLEAR*. Thus, the user can reactivate the output of the received output data to the periphery.

7 Multi Instance Operation

7.1 Features

Multi instance operation is the operation of several communication channels of the same type (C0 and/or C2 communication channels) to the DPV1 slave. Accordingly, a multi-instance functionality exists if in the V1SL configuration file, the number of the selected instances

- *V1SL_CFG_COMPONENT_C0* or
- *V1SL_CFG_COMPONENT_C2*

has a value larger than 1 (refer to Section 25.2).

Therefore each communication channel has to be identified

- By V1SL for input function calls of the user
- By the user for output macro calls of V1SL

To do this, additional call parameters, called handles, are necessary → a handle management at the C0 and C2 firmware interface will expand the DPV1 slave package.

Figure 10 below shows the requirement for the system environment regarding the design of the initialization sequence, and of the initialization elements ‘Detail Info’ and ‘System Info’. The figure represents an expansion of Figure 5.

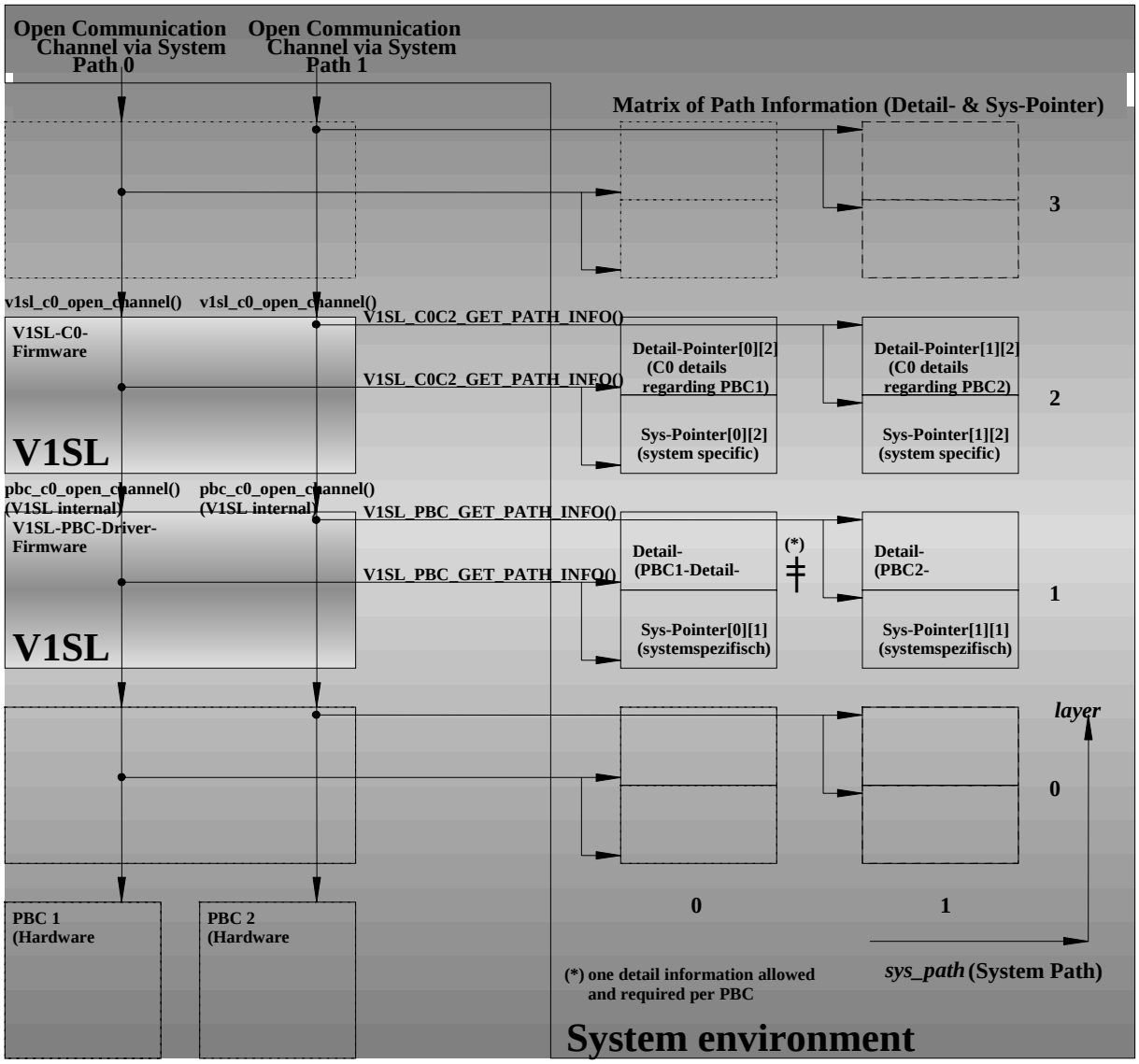


Figure 10: System Paths (*sys_path*) through the Layer Stack during Multi-Instance mode

7.2 Preconditions

The system environment and the user have to meet the following requirements for multi-instance operation:

- Several PBCs have to be contained in the slave module (refer to Section 'Multi-Device Operation' below), since per PBC, one C0 and one C2 communication channel can be operated as a maximum .
- Expansion of the user functions/macros with a handle management.

7.3 Description

In the description of the input functions and output macros of the C0 and C2 firmware in Section 10, multi-instance capability is not initially discernable. Therefore, special subsections 10.1.6, and 10.2.6 include additions to the description of the input functions and output macros at the standard interface of the V1SL that refer to this attribute of the C0 and C2 firmware.

7.4 Activation

The request for multi instance operation is recognized and activated automatically by the V1SL firmware package, based on the conditions mentioned.

Note: Separate activation of the multi-instance capability for C0 and C2 firmware is not possible.

8 Multi-Device Operation

8.1 Features

In multi-device operation, several PBCs are used to operate as slaves. Accordingly, multi-device functionality exists if in the V1SL configuration file, the constant

- *V1SL_CFG_COMPONENT_DPC31*

has a value larger than 1 (refer to Section 25.1)

The result is that each PBC has to be identified

- by the PBC driver for input function calls of the system environment
- by the system for output macro calls of the PBC driver

To do this, additional call parameters, called handles, are necessary → the PBC driver firmware will be expanded with a handle management at the PBC driver firmware interface.

Figure 9 below shows the requirement for the system environment regarding the design of the initialization sequence, and the initialization elements 'Detail Info' and 'System Info'. The figure represents an expansion of Figure 3.

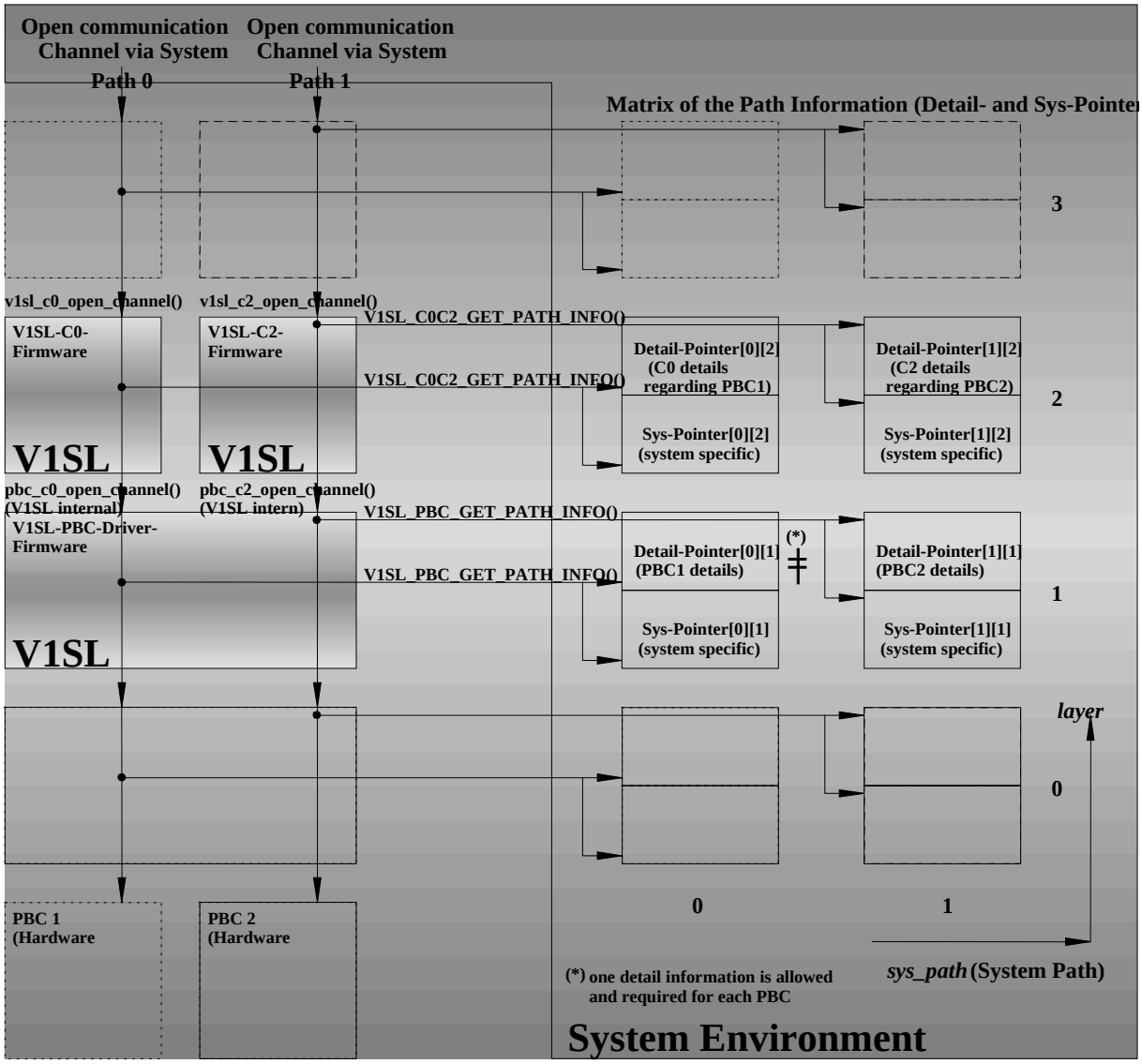


Figure 11: System Paths (*sys_path*) through the Layer Stack in Multi-Device Operation

8.2 Preconditions

The system environment has to meet the following requirements for multi-device operation:

- Several PBCs have to be contained in the slave module.
- Expansion of the system functions with a handle management.

8.3 Description

In the description of the input functions of the PBC driver firmware in Sections 17 and 18, multi-device capability is not initially discernable. Therefore, the special subsection 17.8 includes additions to the description of the input functions and output macros of the PBC driver.

8.4 Activation

The request for multi device operation is recognized and activated automatically by the V1SL firmware, based on the conditions mentioned.

9 V1SL Interface to the System

9.1 Generally Used System Interface Functions/Macros

9.1.1 Generally Used System Interface Input Functions

9.1.1.1 Overview

Input Function	Description
v1sl_init	Initializing the V1SL
v1sl_get_version	Read the version number and the generated components of the V1SL firmware

9.1.1.2 Initialization of the V1SL Firmware

Prototype:

```
void V1SL_SYS_CODE_ATTR v1sl_init (void)
```

This function has to be called **once** by the system environment as the first function of the V1SL firmware package, after switching on the slave module. This initializes internal variables that are needed for further operation. The function is completed with the return to the initiator. The initiator does not receive an explicit acknowledgement.

Note: After the V1SL has been initialized, operation is not yet possible. In addition it is necessary to set up one or several communication channels to the slave instances by calling *v1sl_c0_open_channel()* or *v1sl_c2_open_channel()*.

Input Function:		**v1sl_init**
Meaning:		Initialize V1SL firmware
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range		Meaning
Corresponding Output Macros:		

9.1.1.3 Read the Version Number and the Generated Components of the V1SL Firmware

Prototype:

```
void V1SL_SYS_CODE_ATTR v1sl_get_version (V1SL_SYS_VERSION_PTR
version_ptr)
```

If the system environment calls this function, the version number and the generated components of the V1SL can be read. The call can be made any time. The function is

completed with the return to the initiator. The initiator does not receive an explicit acknowledgement.

Input Function:		**v1sl_get_version**
Meaning:	Read version number and the generated components of the V1SL firmware	
Transfer:		
Parameter	Value Range	Meaning
version_ptr	(refer to Section 11.1.8 'V1SL')	Pointer to the version structure, which contains information after the return to the initiator
Return:		
Value Range	Meaning	
Corresponding Output Macros:		

9.1.2 Generally Used System Interface Output Macros

9.1.2.1 Overview

Output Macros	Description
V1SL_ENTER	Enter a protected program segment
V1SL_EXIT	Exit a protected program segment
V1SL_FATAL_ERROR	Display of a fatal error

9.1.2.2 Disable Calls of the V1SL Firmware

Prototype:

```
#define V1SL_ENTER(_SYS_PTR)
```

When this output macro is called, the V1SL enters a non-interruptible segment. This avoids calling V1SL input functions during critical actions that would have an interfering effect on the state machines.

Exception: If the V1SL indicates events to the user while a non-interruptible segment is being processed, the user can call input functions of the V1SL while processing the events. In this case, the output macro pair for disabling and enabling has to be laid out in a multi-stage mode.

It is possible to disable the three main modules of V1SL separately. Using the parameter *_SYS_PTR*, the system environment can determine which module called the disable macro, and derive from this which function calls are no longer allowed (refer also to Section 15.4)

- C0 Firmware:
  - *v1sl_c0_...()*
  - *v1sl_al_...()*
  - *v1sl_c1_...()*
  - *v1sl_c0c1_perform_services()*
- C2 Firmware:
  - *v1sl_c2_...()*

- ♦ *v1sl_c2_perform_services()*
- ♦ V1SL system functions *v1sl_upper_...()*

- PBC driver:
  - ♦ All functions of the C0- and C2 firmware

Output Macro:		**V1SL_ENTER**
Meaning:	Disable the calls of the V1SL firmware	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_SYS_PTR	V1SL_SYS_SYSTEM_PTR	System pointer of the communication channel whose request is just being processed. The V1SL fills in this data; the system environment can optionally ignore it, or use it for different identification purposes.
Recommended Actions:		
•		
Corresponding Input Functions:		

9.1.2.3 Enable Calls of the V1SL Firmware

Prototype:

```
#define V1SL_EXIT()
```

By calling this output macro, the V1SL exits a non-interruptible segment.

Output Macro:		**V1SL_EXIT**
Meaning:	Enable calls of the V1SL firmware	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_SYS_PTR	V1SL_SYS_SYSTEM_PTR	System pointer of the communication channel whose request is just being processed. The V1SL fills in this data; the system environment can optionally ignore it or use it for different identification purposes.
Recommended Actions:		
•		
Corresponding Input Functions:		

9.1.2.4 Indicate Fatal V1SL Error

Prototype:

```
#define V1SL_FATAL_ERROR(_ERROR_DETAIL_PTR)
```

By calling this output macro, the V1SL informs the system environment of the occurrence of a fatal error. In the case of errors of this type, the V1SL stores the cause for the error locally. Only the cause that initially triggered the error is entered. Subsequent errors do not cause additional calls of the output macros.

It is recommended to stop further processing of the firmware in the module immediately, and not to return from the error handling routine. However, the V1SL is also able to operate in system environments where this is not possible. In this case, the user has to select the switch

- V1SL_CFG_ENVIRONMENT_CONTINUE_ON_FATAL_ERROR

in the V1SL configuration file (refer to Section 25.1)

Attention: After a fatal error has been signalled, the V1SL no longer processes functions. The firmware has to be restarted by calling *v1sl_init()*. Errors that require calling this output macro usually need to be remedied within the V1SL firmware package.

Output Macro:		**V1SL_FATAL_ERROR**
Meaning:	Indicate a fatal error of V1SL	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_ERROR_DETAIL_PTR	V1SL_FAR_ERROR_PTR / NIL	Pointer to the error data block of the DPV1 slave: • No information available in the local error block • Otherwise, pointer to local error block with detailed error description
Recommended Actions:		
• Reinitialize DPV1 slave, • Inform firmware development (refer to cover sheet of the document)!		
Corresponding Input Functions:		
All input functions		

9.2 System Interface Functions/Macros Used by C0/C2 Firmware

9.2.1 System Interface Input Functions Used by the C0/C2 Firmware

9.2.1.1 Overview

Input Function	Description
v1sl_c0c1_perform_services	Processing the PBC events for the C0 firmware
v1sl_c2_perform_services	Processing the PBC events for the C2 firmware

9.2.1.2 Process PBC Events for the C0 Firmware

Prototype:

```
void V1SL_SYS_CODE_ATTR v1sl_c0c1_perform_services (Unsigned16
event_bit_field)
```

The system environment has to call this function for the C0 firmware as a result of the occurrence of PBC events (refer to section 18.2).
If needed, a context change between the priority layers of the PBC driver and the C0 firmware can be executed (refer to 15.4).

Input Function:		v1sl_c0c1_perform_services
Meaning:		Process PBC events for the C0 firmware
Transfer:		
Parameter	Value Range	Meaning
event_bit_field		Event bit field which was fetched during processing the macro *PBC_C0C1_EVENT_INDICATION()*
Return:		
Value Range		Meaning
Corresponding Output Macros:		
PBC_C0C1_EVENT_INDICATION()		

9.2.1.3 Process the PBC Events for the C2 Firmware

Prototype:

```
void V1SL_SYS_CODE_ATTR v1sl_c2_perform_services (Unsigned16
event_bit_field)
```

The system environment has to call this function for the C2 hardware as a result of the occurrence of PBC events (refer to section 18.3).
If needed, a context change between the priority layers of the PBC driver and the C2 firmware can be executed (refer to 14.4).

Input Function:		v1sl_c2_perform_services
Meaning:		Process PBC events for the C2 firmware
Transfer:		
Parameter	Value Range	Meaning
event_bit_field		Event bit field which was fetched during processing the macro *PBC_C2_EVENT_INDICATION()*
Return:		
Value Range		Meaning
Corresponding Output Macros:		
PBC_C2_EVENT_INDICATION()		

9.2.2 Interface Expansion of the Input Functions in Multi Instance Operation

When multi instance operation is recognized, a parameter *handle* is added to the following input functions of the C0 and C2 firmware, which is used as a reference to the communication channel.

Prototype:

```
void V1SL_SYS_CODE_ATTR v1sl_c0c1_perform_services (..., Unsigned8
handle)
```

```
void V1SL_SYS_CODE_ATTR v1sl_c2_perform_services (..., Unsigned8
handle)
```

Input Function:		v1sl_c0c1_perform_services/v1sl_c2_perform_services
Meaning:		Process PBC events for the C0 and C2 firmware
Transfer:		
Parameter	Value Range	Meaning
...		Parameters that depend on the concrete function
handle		Handle of the communication channel
Return:		
Value Range		Meaning
Corresponding Output Macros:		
PBC_C0C1_EVENT_INDICATION(), PBC_C2_EVENT_INDICATION()		

9.2.3 System Interface Output Macros Used by C0/C2 Firmware

9.2.3.1 Overview

Output Macros	Description
V1SL_C0C2_GET_PATH_INFO	Request for the path information of a C0/C2 commmunication channel
V1SL_C0C2_RELEASE_PATH_INFO	Release of the path information of a C0/C2 communication channel

9.2.3.2 Request the Path Information of a C0/C2 Communication Channel

Prototype:

```
#define V1SL_C0C2_GET_PATH_INFO(_RETURN_VALUE,_SYS_PATH,
_SYS_PTR_PTR,_DETAIL_PTR_PTR)
```

According to the LSA model, the output macro determines two pointers from the specified path description (*_SYS_PATH)*:

- The system pointer is used to identify the layer below associated with this path. It is not needed by V1SL, but it is stored, and transferred to the system environment as a parameter for some output macros.
- The detail pointer points to the information, which contains the specifics of the concrete implementation regarding protocol processing. The structures of the detail information that is valid for the C0 and C2 firmware are provided in the sections 11.2.1 'C0 Detail Info Structure and Pointer' and 11.3.1 '. The V1SL expects different detail information structures depending on whether the specified path is to be used for communication with the C0 or C2 firmware. The system component specifies the values of the individual structural elements.

The parameter *_SYS_PATH* is specified for the C0 and C2 firmware by their users via the functions *v1sl_c0_open_channel()* and *v1sl_c2_open_channel()*.

Output Macro:		V1SL_C0C2_GET_PATH_INFO
Meaning:		Request the path information of a C0/C2 communication channel
Transfer:		
Parameter	Type, Attribute / Value Range	Meaning
_RETURN_VALUE	Unsigned16 / V1SL_SYS_PATH_OK	Status of operation (the V1SL does not fill in this data; it is to be assigned by the system environment): • System and detail information was entered in the transferred pointers to be evaluated by the C0/C2 firmware .

		• The V1SL interprets all other values as an error.
_SYS_PATH	V1SL_SYS_PATH_TYPE	Communication path information; used by the system environment to provide the system and detail information available to the C0 and C2 firmware; the V1SL fills in this data.
_SYS_PTR_PTR	V1SL_SYS_SYSTEM_PTR V1SL_INT_DATA_ATTR *	The V1SL transfers the address of a pointer; its content is to be assigned by the system environment in the case of a positive acknowledgement. If the system environment does not need the data when calling output macros, NIL can be assigned to it.
_DETAIL_PTR_PTR	V1SL_SYS_C0_DETAIL_PTR V1SL_INT_DATA_ATTR * or V1SL_SYS_C2_DETAIL_PTR V1SL_INT_DATA_ATTR *	The V1SL transfers the address of a pointer. In the case of a positive acknowledgement, the system environment fills in the detail information of the communication path of the C0/C2 firmware.
Recommended Actions:		
•		
Corresponding Input Functions:		

9.2.3.3 Release the Path Information of a C0/C2 Communication Channel

Prototype:

```
#define V1SL_C0C2_RELEASE_PATH_INFO(_SYS_PTR,_DETAIL_PTR)
```

By calling this macro, the C0/C2 firmware returns the path description, previously determined with the output macro *V1SL_C0C2_GET_PATH_INFO()*, to the system environment in the form of two pointers (system pointer and detail pointer).

Output Macro:	**V1SL_C0C2_RELEASE_PATH_INFO**	
Meaning:	Release the path information of a C0/C2 communication channel	
Transfer:		
Parameter	Type, Attribute / Value Range	Meaning
_SYS_PTR	V1SL_SYS_SYSTEM_PTR	System pointer; the V1SL fills in this data
_DETAIL_PTR	V1SL_SYS_C0_DETAIL_PTR or V1SL_SYS_C2_DETAIL_PTR	Detail pointer of the C0/C2 firmware; the V1SL fills in this data
Recommended Actions:		
•		
Corresponding Input Functions:		

10 V1SL Standard Interface to the User

10.1 Input Functions

10.1.1 Overview

C0 Input Functions	Description
v1sl_c0_open_channel	Opens a C0 communication channel
v1sl_c0_close_channel	Closes a C0 communication channel
v1sl_c0_add	Sets up the memory resources needed by the slave
v1sl_c0_withdraw	Releases the memory resources of the slave
v1sl_c0_control	Controls the slave
v1sl_c0_get_real_cfg_ptr	Fetches pointer to expected configuration data buffer
v1sl_c0_real_cfg_update	Makes expected configuration data available
v1sl_c0_calc_in_out_len	Calculates the input and output data length for user data
v1sl_c0_get_input_ptr	Fetches pointer to current input data buffer
v1sl_c0_input_update	Makes input data available
v1sl_c0_get_output_info	Fetches pointer to current output data buffer and status of the output data buffer
v1sl_c0_set_diag	Makes diagnostic data available
AL Input Functions	**Description**
v1sl_al_set_alarm	Sets alarms
v1sl_al_withdraw_alarm	Cancels alarms
C1 Output Macros	**Description**
v1sl_c1_read_ds_done	Signals completion of service 'Read Data Set' via the C1 firmware
v1sl_c1_write_ds_done	Signals completion of service 'Write Data Set' via the C1 firmware
C2 Input Functions	**Description**
v1sl_c2_open_channel	Opens a C2 communication channel
v1sl_c2_close_channel	Closes a C2 communication channel
v1sl_c2_initiate_done	Responds to an Initiate PDU
v1sl_c2_data_transport_done	Signals completion of the service 'Data Transport'
v1sl_c2_read_ds_done	Signals completion of service 'Read Data Set' via the C2 firmware
v1sl_c2_write_ds_done	Signals completion of service 'Read Data Set' via the C2 firmware
v1sl_c2_user_abort	Connection is canceled by the user

10.1.2 Input Functions of C0 at the User Interface

10.1.2.1 Open a C0 Communication Channel

Prototype:

```
void V1SL_IFA_CODE_ATTR v1sl_c0_open_channel (SYS_PATH_TYPE sys_path)
```

The user opens a communication channel to the C0 firmware. This includes allocating and initializing resources that the V1SL C0 component needs. In addition, the communication channel to the layer below (PBC driver) is set up.

After completion of the function the user is informed via the output macro *V1SL_C0_OPEN_CHANNEL_DONE()* in an asynchronous way.

Note: After establishment of a communication channel, operation is not yet possible. In addition, it is mandatory to initialize the slave instance with parameters of the needed memory resources by calling *v1sl_c0_add()*.

Calling the function is possible only after the V1SL has been initialized (input function *v1sl_init()*).

Input Function:		**v1sl_c0_open_channel**
Meaning:		Open a C0 communication channel
Transfer:		
Parameter	Value Range	Meaning
sys_path		System path: ID of a C0 communication channel assigned by the system . The concrete type of this parameter is specified by the system (refer to parameter *V1SL_SYS_PATH_TYPE* in Section 25. 'Filling in the File 'v1sl_cfg.h' by the user').
Return:		
Value Range		Meaning
Corresponding Output Macros:		
V1SL_C0_OPEN_CHANNEL_DONE()		

10.1.2.2 Close a C0 Communication Channel

Prototype:

```
void V1SL_IFA_CODE_ATTR v1sl_c0_close_channel (void)
```

The user closes a communication channel to the C0 firmware. The allocated resources are released. In addition, the communication channel to the layer below (PBC driver) is closed.
After the completion of the function the user is informed via the output macro *V1SL_C0_CLOSE_CHANNEL_DONE()* in an asynchronous way.

This function can only be called after the C0 communication channel was opened (input function *v1sl_c0_open_channel()*).

Input Function:		**v1sl_c0_close_channel**
Meaning:		Clos a C0 communication channel
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range		Meaning
Corresponding Output Macros:		
V1SL_C0_CLOSE_CHANNEL_DONE()		

10.1.2.3 Set Up the Slave Memory resources

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c0_add (V1SL_IFA_C0_PARAMETER_PTR
para_ptr)
```

By calling this function, the user sets the memory resources of the C0 slave instance and decides on the services to be used:

- AL - services for alarm handling
- C1 - services for acyclic read and write of data sets

The parameters are transferred to the layer below (PBC driver), and the slave-specific receive and send buffers are allocated.

As a result of the function, the user is informed on the slave asynchronously via the output macros below:

- *V1SL_C0_WD_STATE_REPORT()* - current state of DP watchdog
- V1SL_C0_DP_STATE_REPORT() - current DP state
- *V1SL_C0_LED_STATE_REPORT()* - current state of bus error LED

In addition, the V1SL indicates to the user the availability of a configuration buffer for expected configuration data.
(output macro *V1SL_C0_REAL_CFG_BUFFER_CHANGED()*)

This function can be called only after the C0 communication channel has been opened (input function *v1sl_c0_open_channel()*).

Note: Operation is not yet possible after setting up the slave memory resources. In addition, it is mandatory to provide the slave instance with a current expected configuration by calling *v1sl_c0_real_cfg_update()*, and starting it with *v1sl_c0_control()*.

Input Function:		**V1sl_c0_add**
Meaning:	Set up a slave memory resources	
Transfer:		
Parameter	Value Range	Meaning
para_ptr	(refer to Section 11.2.2 'C0 Parameter Structure and Pointer' on page 101)	Describing the memory resources of the C0 slave instance
Return:		
Value Range	Meaning	
V1SL_OK V1SL_ERR_SEQUENCE V1SL_ERR_PARAMETER	• Execution OK • Communication channel not opened, or function already executed • Wrong slave parameters	
Corresponding Output Macros:		
V1SL_C0_WD_STATE_REPORT(), V1SL_C0_DP_STATE_REPORT(), V1SL_C0_LED_STATE_REPORT(), V1SL_C0_REAL_CFG_BUFFER_CHANGED()		

10.1.2.4 Release Slave Memory resources

Prototype:

Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c0_withdraw (void)

By calling this function, the memory resources of a C0 slave instance are released. This refers to all receive and send data buffers allocated during the set up of this instance. Also, the V1SL returns user diagnostic buffers which may be still in the slave (*V1SL_C0_DIAG_CHANGED()*).

The user has to stop the slave first (*v1sl_c0_control()*). The slave is stopped if the transition to the DP state *V1SL_DP_STATE_OFF* was signaled to the user via *V1SL_C0_DP_STATE_REPORT()*. In addition, the user has to cancel all alarms (*v1sl_al_withdraw_alarm()*) and terminate current data set services via the C1 firmware (*v1sl_c1_read_ds_done()* and *v1sl_c1_write_ds_done()*).
After the completion of the function *v1sl_c0_withdraw(),* the user is informed via the output macro *V1SL_C0_WITHDRAW_DONE()* in an asynchronous way.
This function can be called only after the set up of the C0 slave memory resources (input function *v1sl_c0_add()*).

Input Function:		**v1sl_c0_withdraw**
Meaning:		Release memory resources of the C0 slave instance
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range		Meaning
V1SL_OK_ASYNC V1SL_ERR_SEQUENCE V1SL_ERR_DP_STATE V1SL_ERR_SSA_STATE		• Execution OK • Communication channel not open, or memory resources not set up • Slave is not in the state *V1SL_DP_STATE_OFF* • A request for setting a new slave address has not been completed (output macro *V1SL_NEW_SSA()*)
Corresponding Output Macros:		
V1SL_C0_WITHDRAW_DONE()		

10.1.2.5 Control the Slave

Prototype:

void V1SL_IFA_CODE_ATTR v1sl_c0_control (Unsigned8 mode)

By calling this function, the user can control the state of the slave. This is necessary in the following cases:

- The user wants to start operation of the slave after establishment of a communication channel and setting up the C0 slave's memory resources.
- The user wants to stop the slave because of internal events, or prior to releasing the C0 slave's memory resources.
- The user informs the slave that the processing of new data is finished. This applies to:
  - Slave station address data (*V1SL_C0_NEW_SSA()*)
  - Parameterization data (*V1SL_C0_NEW_PRM()*)
  - Configuration data sent by the master (*V1SL_C0_NEW_CFG()*)
- The user wants to reset the slave because of internal events, so that it has to be reparameterized by a DP(V1) master.

The user wants to state that it is ready for data exchange with the parameterization master after finishing its internal set up (parameterization, configuring, and module parameterization by means of parameterization data sets) is completed (refer to section (refer to Section 6.5)
There is no acknoweldgement from the slave. The slave‘s reactions are stated to the user asynchronously via output macros (for example, *V1SL_C0_DP_STATE_REPORT()*).

The function can be called if the C0 slave’e memory resources is set up ((*v1sl_c0_add()*).

Input Function:		**v1sl_c0_control**
Meaning:	Control the slave	
Transfer:		
Parameter	Value Range	Meaning
mode	(refer to Section 12.2.1 on page 117)	Control Parameters:
	V1SL_CONTROL_ START	• With this command, the slave changes from the DP mode *V1SL_DP_STATE_OFF* to the DP mode *V1SL_DP_STATE_NO_DATA_EX*, and can then be parameterized by the master. This command has to be performed after establishment of a communication channel and setting up the C0 slave’s memory resources. The call is possible only in the DP mode *V1SL_DP_STATE_OFF* .
	V1SL_CONTROL_ STOP	• With this command, the slave enters the DP mode *V1SL_DP_STATE_OFF*, and can then no longer be parameterized by any master. A restart can be made via the slave control, using *V1SL_CONTROL_START* . The call is possible only in the DP mode *V1SL_DP_STATE_NO_DATA_EX* or *V1SL_DP_STATE_DATA_EX*. With this call, parameterization data (*V1SL_C0_NEW_PRM()*) and configuration data sent by the master (*V1SL_C0_NEW_CFG()*), that may be processed right now, are no longer valid for the user (no more accesses to the buffers). Also an acknowledgement to the slave regarding this data is no longer allowed.
	V1SL_CONTROL_ LEAVE_MASTER	• With this command, the user can, if needed, reset an active slave into the DP mode *V1SL_DP_STATE_NO_DATA_EX* (for example, if the configuration changes). With it, the slave requests reparameterization by the master. The call is possible only in the DP mode *V1SL_DP_STATE_NO_DATA_EX* or *V1SL_DP_STATE_DATA_EX* .
	V1SL_CONTROL_ SSA_DONE	• With this command, the user informs the slave that processing the ‘Set Slave Address‘ telegram is completed (*V1SL_C0_NEW_SSA()*).
	V1SL_CONTROL_ PRM_OK	• With this command, the user informs the slave that processing the parameterization data was completed successfully. (*V1SL_C0_NEW_PRM()*).
	V1SL_CONTROL_ PRM_ERROR	• With this command, the user rejects a wrong parameterization telegram (*V1SL_C0_NEW_PRM()*).
	V1SL_CONTROL_ CFG_OK	• With this command, the user informs the slave that the comparison of received configuration data sent by the master with its expected configuration (*V1SL_C0_NEW_CFG()*) was successful.
	V1SL_CONTROL_ CFG_UPDATE	• With this command, the user informs the slave that the comparison of received configuration data sent by the master with its expected configuration was successful.(*V1SL_C0_NEW_CFG()*). At the same time it wants to make the configuration data sent by the master message available as the expected configuration for reading by other bus stations.

	V1SL_CONTROL_CFG_ERROR V1SL_CONTROL_APP_READY	• With this command, the user rejects a wrong configuration telegram sent by the master (*V1SL_C0_NEW_CFG()*). • With this command, the user informs the slave that it is ready to receive user data. The call always has to be made after the receipt of a configuration telegram sent by the master in the DP mode *V1SL_DP_STATE_NO_DATA_EX* . As long as the call is not made but the DP mode *V1SL_DP_STATE_DATA_EX* has been reached, the slave signals static diagnostic to the parametrization master (refer to Section 6.5) 'Slave Control by means of 'Application Ready".

Return:	
Value Range	Meaning
V1SL_OK	• Execution OK
V1SL_ERR_SEQUENCE	• Command not allowed for current slave mode
V1SL_ERR_PARAMETER	• Impermissible control parameter
V1SL_ERR_DP_STATE	• Command not allowed for current DP mode
V1SL_ERR_REAL_CFG_STATE	• The user has not provided expected configuration data yet
V1SL_ERR_SSA_STATE	• Acknowledgement for processing new slave address data is outstanding, or no slave address data was received
V1SL_ERR_TARGET_CFG_STATE	• No configuration data received from the master
V1SL_ERR_APP_STATE	• The user has already signalled that it is ready for data exchange
Corresponding Output Macros:	
V1SL_C0_DP_STATE_REPORT(), V1SL_C0_WD_STATE_REPORT(), V1SL_C0_LED_STATE_REPORT()	

10.1.2.6 Fetch Pointer to Expected Configuration Data Buffer

Prototype:

```
V1SL_LL_UNSIGNED8_PTR V1SL_IFA_CODE_ATTR   v1sl_c0_get_real_cfg_ptr
(void)
```

By calling this function, the user receives a pointer to the expected configuration data buffer. Then, the user can copy the expected configuration data to the memory area referenced by the pointer. The user receives the buffer pointer synchronously; there is no explicit acknowledgement via an output macro.

This function can be called after the user was informed that a free buffer for the expected configuration data (*V1SL_C0_REAL_CFG_BUFFER_CHANGED()*) is available.

Input Function:		**v1sl_c0_get_real_cfg_ptr**
Meaning:		Fetch pointer to expected configuration data
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range		Meaning
ptr NIL		• Pointer to the current buffer for expected configuration data • No buffer available **The user is not allowed to copy data to this address!**
Corresponding Output Macros:		

10.1.2.7 Provide Expected Configuration Data

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c0_real_cfg_update (Unsigned8
real_cfg_len)
```

By calling this function, the user's current expected configuration is routed to the layer below (PBC driver). After that, the parameterization master or a Class 2 master can read the expected configuration data. This action is not acknowledged to the user.

Note: The user must perform this function prior to starting the slave for the first time (*v1sl_c0_control()*), so that the activated slave returns valid configuration data.

This function can be called after the user has fetched the pointer to a free buffer for expected configuration data, and has copied its data to this area (input function *v1sl_c0_get_real_cfg_ptr()*).

Input Function:		**v1sl_c0_real_cfg_update**
Meaning:		Provide expected configuration data
Transfer:		
Parameter	Value Range	Meaning
real_cfg_len	001..c0_cfg_buffer_len (refer to Section 11.2.2 'C0 Parameter Structure and Pointer ' on page 101)	Length of expected configuration data (in bytes)
Return:		
Value Range		Meaning
V1SL_OK V1SL_ERR_SEQUENCE V1SL_ERR_REAL_CFG_STATE		• Execution OK • Command not allowed for current slave mode • No buffer for expected configuration data was fetched • It should be noted in addtition, that wrong length indications for *real_cfg_len* cause *V1SL_FATAL_ERROR()* !
Corresponding Output Macros:		
V1SL_C0_REAL_CFG_BUFFER_CHANGED()		

10.1.2.8 Calculate the Input and Output Data Length for User Data

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c0_calc_in_out_len
(V1SL_IFA_IN_OUT_CALC_PTR dat_ptr)
```

By calling this function, the user can calculate the length of the input and output data area for user data. The calculation can be based on any data field with configuration IDs (refer to Section 6.3).

The use of this function by the user is seen primarily in conjunction with the evaluation of received configuration data sent by the master (*V1SL_C0_NEW_CFG()*). The user

receives the calculated data lengths synchronously. There is no explicit acknowledgement via an output macro.

This function can be called if the C0 slave's memory resources have been set up.

Input Function:		**v1sl_c0_calc_in_out_len**
Meaning:	Calculate the input and output data length for user data	
Transfer:		
Parameter	Value Range	Meaning
dat_ptr	(refer to Section 11.2.5 'C0 Input/Output Data Length Structure and Pointer ' on page 104)	Pointer to a union that contains information of the configuration data that is to be evaluated.
Return:		
Value Range	Meaning	
V1SL_OK	• Execution OK • The union which the user transferred at the call contains the calculated lengths (in bytes) for input and output data	
V1SL_ERR_SEQUENCE	• Command not allowed for current slave mode	
V1SL_ERR_CFG_DATA	• The configuration data is incorrect, or it exceeds the maximum user data length specified by the user via *v1sl_c0_add()*.	
Corresponding Output Macros:		

10.1.2.9 Fetch Pointer to Current Input Data Buffer

Prototype:

```
V1SL_LL_UNSIGNED8_PTR V1SL_IFA_CODE_ATTR v1sl_c0_get_input_ptr (void)
```

By calling this function, the user receives the pointer to a free input data buffer of the slave. The length of the buffer corresponds to the length requested in the configuration telegram sent by the master (refer to Section 10.1.2.8 'Calculating the Input and Output Data Length for User Data'). The user can copy the input data to the memory area referenced by the pointer. The buffer pointer is received synchronously; there is no explicit acknowledgement via an output macro.

This function can be called after the C0 slave's memory resources have been set up.

Input Function:		**v1sl_c0_get_input_ptr**
Meaning:	Fetch pointer to current input data buffer	
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range	Meaning	
ptr	• Pointer to current input data buffer	
NIL	There is no pointer for the input data. **The user is not allowed to copy data to this address!** The reason for this may be one of the following: • The user did not positively acknowledge the comparison of configuration data sent by the master and its expected configuration (*v1sl_c0_control()*). • The number of inputs resulting from the configuration sent by the master is 0.	
Corresponding Output Macros:		

10.1.2.10 Provide Input Data

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c0_input_update (void)
```

By calling this function, the current input data buffer of the user is routed to the layer below, the PBC driver, and thus to the PBC itself. Then, the parameterization master can read the input data. The service is processed synchronously; there is no explicit acknowledgement via an output macro.

Note: If inputs exist (according to the configuration sent by the master), the function is to be called immediately after the user confirms a correct configuration sent by the master with *v1sl_c0_control()*. This requirement is a precondition for the slave entering the DP mode *V1SL_DP_STATE_DATA_EX*. This ensures that with the first data exchange, current input data is transmitted to the parameterization master.

The function can be called only after the user has fetched the pointer to a free input data buffer, and has copied input data to this area (input function *v1sl_c0_get_input_ptr()*).

Input Function:		**v1sl_c0_input_update**
Meaning:		Provide input data
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range		Meaning
V1SL_OK V1SL_ERR_SEQUENCE V1SL_ERR_INPUT_STATE		• Execution OK • Command not allowed for current slave mode • No input data buffer was fetched.
Corresponding Output Macros:		

10.1.2.11 Fetch Pointer to the Current Output Data Buffer and Status

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c0_get_output_info
(V1SL_IFA_OUTPUT_INFO_PTR output)
```

By calling this function, the user gets a pointer to the current output data buffer of the parameterization master. In addition, the function provides information about the status of the data in this buffer, and about the general conditions that may be important to the evaluation of the data:

- The length of the data buffer has is or is not equal to 0
- The data buffer contains new data that the user has not processed yet
- The data buffer contains data that was deleted
- Information on a currently received global control telegram

The service is processed synchronously; there is no explicit acknowledgement via an output macro.

The function can be called if the C0 slave's memory resources are set up.

Input Function:		**v1sl_c0_get_output_info**
Meaning:	Fetch pointer to current output data buffer and status	
Transfer:		
Parameter	Value Range	Meaning
output	(refer to Section 11.2.6 'C0 Output Data Info Structure and Pointer')	Pointer to a structure where the V1SL enters the current output data buffer and its status
Return:		
Value Range	Meaning	
V1SL_OK	• Execution OK • The pointer to the current output data buffer and its status are stored in the transferred structure (elements *output→ptr*, *output→state*)	
V1SL_ERR_SEQUENCE	• Command not allowed for current slave mode	
Corresponding Output Macros:		

10.1.2.12 Make Diagnostic Data Available

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c0_set_diag (V1SL_UNION_DIAG_PTR
user_diag, Unsigned8 user_diag_len, Unsigned8 diag_control,
V1SL_C0_USER_ID_TYPE user_id)
```

By calling this function, the user provides diagnostic data to the slave. The diagnostic data is sent at the next possible time as a static part with each diagnostic telegram (refer to figure 7)
The user has to make sure that the buffer size does not exceed the size of the diagnostic buffer that was set when the slave's memory resources were set up:

$$\textbf{Diagnostic Buffer Length} >= \textbf{Length}_{\text{Diag\_User}} + \textbf{Length}_{\text{Alarm\_User}}$$

Note: This function is not suitable for setting alarms.

The user can set up DP diagnostics; the following applies:

- The 6 byte standard diagnostic (refer to EN 50170) is not part of the user diagnostic.
- In DP standard operation, one ID-related, several channel-related, and several device-related diagnostics may be utilized.
- In DPV1 operation, one revision, one ID-related, several channel-related, and several device-related diagnostics may be utilized. Device-related diagnostics are to be encoded as status PDUs.
- The user is responsible for the content of the diagnostic data.

If no error occurs (check return value!), the V1SL keeps the user diagnostic buffer. It is returned to the user asynchronously via the output macro *V1SL_C0_DIAG_CHANGED()*. However, if the user, prior to receiving its diagnostic buffer, wants to make a new diagnostic available, this has to be implemented with a 2nd diagnostic buffer (exchange buffer system). The 1st buffer is returned to the user immediately via the output macro *V1SL_C0_DIAG_CHANGED()*.
Diagnostics are acknowledged (output macro *V1SL_C0_DIAG_FETCHED()*) after the parameterization master has fetched the diagnostic telegram from the slave. This takes place only, if the slave exchanges user data with the parameterization master. The user receives the message about the start of user data exchange by the output macro *V1SL_C0_DATA_EXCHANGE_ACTIVE()*. After the data exchange has been cancelled, no more diagnostic acknowledgements are generated for the user. V1SL signals the cancellation of the data exchange with the output macro *V1SL_C0_DP_STATE_REPORT()*.
The function can be called if the C0 slave's memory resources have been set up.

Input Function:	**v1sl_c0_set_diag**	
Meaning:	Make diagnostic data available	
Transfer:		
Parameter	Value Range	Meaning
user_diag	(refer to Section 11.2.7'	Union that includes the address of the user diagnostic buffer
user_diag_len	000 001..c0_user_diag_buffer_len (refer to Section 11.2.2 'C0')	Length of the user diagnostic buffer: • A previously set user diagnostic is deleted from the slave's diagnostic buffer. Only 6 bytes standard diagnostic and possibly pending alarms are sent in the diagnostic telegram. In this case, the user does not have to transfer a pointer to a diagnostic buffer; refer to previous transfer parameter. • Length of the new user diagnostic data
diag_control	OR-operation of the values (refer to Section 12.2.8 V1SL_EXT_DIAG_SET V1SL_EXT_DIAG_RESET V1SL_EXT_DIAG_UNCHANGE V1SL_EXT_DIAG_OVF_SET V1SL_EXT_DIAG_OVF_RESET V1SL_EXT_DIAG_OVF_UNCHANGE V1SL_STAT_DIAG_SET V1SL_STAT_DIAG_RESET V1SL_STAT_DIAG_UNCHANGE V1SL_SEND_DIAG_WITH_ALARM	Influence on the user diagnostic bits (initial assignment→all bits=0): • Set bit 'extended diagnostic' • Reset bit 'extended diagnostic' • Don't influence bit 'extended diagnostic' • Set bit 'extended diagnostic data overflow' • Reset bit 'extended diagnostic data overflow' • Don't influence bit 'extended diagnostic data overflow' • Set bit 'static diagnostic'; this is possible only in a compatibility mode of the slave (*v1sl_c0_control()*) • Reset bit 'static diagnostic' • Don't influence bit 'static diagnostic' • The diagnostic that is set is sent only together with any alarm; if necessary, the slave waits for the next alarm set by the user. The bit is reset internally by the C0 firmware when leaving the DP mode *V1SL_DP_STATE_DATA_EX* .
user_id		Value that identifies the diagnostic buffer for the user if V1SL signals that the parameterization master fetched a diagnostic (*V1SL_C0_DIAG_FETCHED()*)

Return:	
Value Range	Meaning
V1SL_OK_ASYNC V1SL_ERR_SEQUENCE V1SL_ERR_DIAG_BUFFER	• Execution OK • Command not allowed for current slave mode • Diagnostic elements or diagnostic length wrong • In addition, it should be mentioned that diagnostic length data that exceeds the maximum length which was set causes the call of *V1SL_FATAL_ERROR()*.
Corresponding Output Macros:	
V1SL_C0_DIAG_CHANGED(), V1SL_C0_DIAG_FETCHED()	

10.1.3 Input Functions of AL at the User Interface

10.1.3.1 Set Alarms

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_al_set_alarm (V1SL_IFA_ALARM_PTR
alarm_ptr)
```

By calling this function, V1SL accepts the transferred alarm data. In addition to the net data, the alarm data also includes control information according to the DPV1 specification . The data is transmitted at the next possible time, as changeable (dynamic) part of the diagnostic message (figure 7). The user has to make sure that the buffer size does not exceed the size of the diagnostic buffer that was set when the slave's memory resources were defined.

$$\textbf{Diagnostic Buffer Length} >= \textbf{Length}_{\text{Diag\_User}} + \textbf{Length}_{\text{Alarm\_User}}$$

Specifications:

- When setting alarms, the user has to adhere to the requirements regarding permissible alarm types that he was informed of when the alarm state machine was started (*V1SL_AL_STATE_REPORT()*).
- The number of alarms that are permitted to be processed simultaneously during communication between parameterization master and slave is specified by the type- or sequence mode. It is entirely handled by the V1SL; the user has no influence on it, and can thus set any number of alarms of all permitted types.
- The user is responsible for the content of the alarm data.
- The alarm buffer is to contain only one alarm.

After the alarm is set, the alarm buffer remains with the V1SL. The buffer is returned to the user via the output macro *V1SL_AL_ALARM_ACK()*:

- when the parameterization master acknowledges the alarm or
- when the user cancels the alarm (*v1sl_al_withdraw_alarm()*).

Note: Calling the function *v1sl_al_set_alarm()* in the context of the output macro *V1SL_AL_ALARM_ACK()* is not permitted.

The function can be called only if the slave's alarm state machine is started (output macro *V1SL_AL_STATE_REPORT()*).

Input Function:		v1sl_al_set_alarm
Meaning:	Set the alarm	
Transfer:		
Parameter	Value Range	Meaning
alarm_ptr	(refer to Section 11.2.8)	Pointer to structure with alarm data
Return:		
Value Range	Meaning	
V1SL_OK_ASYNC V1SL_ERR_SEQUENCE V1SL_ERR_AL_STATE V1SL_ERR_QUEUE V1SL_ERR_PARAMETER	• Execution OK • Command not allowed for current slave mode • Alarm state machine not started • Alarm queue is locked; the function was called in the context of the output macro *V1SL_AL_ALARM_ACK()* • The values of the transfer parameters are not in the specified value range • In addition, it should be noted that a wrong length specification of the entire diagnostic causes the call of *V1SL_FATAL_ERROR()*!	
Corresponding Output Macros:		
V1SL_AL_ALARM_ACK()		

10.1.3.2 Withdraw Alarms

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_al_withdraw_alarm (Unsigned8
alarm_type_bit_field, Unsigned8 sequence_number)
```

With this call, the V1SL returns the alarms that were previously set by the user.

The alarms are acknowledged asynchronously within the processing of this function via the output macro *V1SL_AL_ALARM_ACK()*.

The following possibilities for withdrawing alarms are supported:
- All alarms of a type, or of several types, and/or
- All alarms with the specified sequence number, or with all sequence numbers.

Note: The user has to call the function after closing the alarm state machine (*V1SL_AL_STATE_REPORT()*), in order to withdraw **all** alarms. This is the precondition for restarting the alarm state machine, or for shutting down the slave (input function *v1sl_c0_withdraw()*).

The function can be called only if the C0 slave's memory resources are set up.

Input Function:		v1sl_al_withdraw_alarm
Meaning:	Withdraw alarms	
Transfer:		
Parameter	Value Range	Meaning
alarm_type_bit_field	The parameter results out of the OR operation of the specified values (refer to Section 12.2.10 'Alarm') V1SL_ALARM_TYPE_ DIAG_VALUE V1SL_ALARM_TYPE_ PROC_VALUE	Bit field of the alarms that are to be withdrawn: • Diagnostic alarm • Process alarm

	V1SL_ALARM_TYPE_ PULL_VALUE V1SL_ALARM_TYPE_ PLUG_VALUE V1SL_ALARM_TYPE_ STAT_VALUE V1SL_ALARM_TYPE_ UPDT_VALUE V1SL_ALARM_TYPE_ MANU_VALUE As an alternative, one of the following values can be used (without OR operation with other values!) V1SL_ALARM_TYPE_ NONE_VALUE V1SL_ALARM_TYPE_ ALL_VALUE	• Pull alarm • Plug alarm • Status alarm • Update alarm • All manufacturer-specific alarm types • No alarm • All alarm types
sequence_number	(refer to Section 12.2.10 'Alarm') V1SL_SEQUENCE_NUMBER_MIN ... V1SL_SEQUENCE_NUMBER_MAX As an alternative, the following value can be used V1SL_SEQUENCE_NUMBER_ALL	Sequence number of the alarms to be withdrawn: • Alarms with the specified sequence number are withdrawn • Alarms of all sequence numbers are withdrawn
Return:		
Value Range		Meaning
V1SL_OK V1SL_ERR_SEQUENCE		• Execution OK • Command not allowed for current slave mode
Corresponding Output Macros:		
V1SL_AL_ALARM_ACK()		

10.1.4 Input Functions of C1 at the User Interface

10.1.4.1 Signal the Completion of the Service ‘Read Data Set’ via C1 Firmware

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c1_read_ds_done (void)
```

By calling this function, the user signals to the slave that a previously requested ‘Read Data Set’ service (output macro *V1SL_C1_READ_DS()*) is completed.
The response data is transferred to the buffer which the slave used for the request. At the same time, the write authorization for this data buffer passes to the slave.
In the case of a positive acknowledgement, the reply data is to be structured according to a Read-RES-PDU, and in the case of a negative acknowledgement, according to a Read-NRS-PDU.

Note: If, prior to acknowledging the processing of the data set, the slave exits the DP mode *V1SL_DP_STATE_DATA_EX*, this function is to be called nevertheless. This is the precondition for restarting the C1 state

machine, or for shutting down the slave (input function *v1sl_c0_withdraw()*).

This function can be called only after previously indicating the service ‘Read Data Set’ via the output macro *V1SL_C1_READ_DS()*, and only in the DP mode *V1SL_DP_STATE_DATA_EX*.

Input Function:		**v1sl_c1_read_ds_done**
Meaning:		Signal the completion of the service 'Read Data Set' via the C1 firmware
Transfer:		
Parameter	Value Range	Meaning
Structure of the Data Buffer:		
positive acknowledgement		_READ_DS_PTR→res (refer to Section 11.4.1'C1/C2 DS_READ')
negative acknowledgement		_READ_DS_PTR→nrs (refer to Section 11.4.1 'C1/C2 DS_READ'))
Return:		
Value Range		Meaning
V1SL_OK V1SL_ERR_SEQUENCE V1SL_ERR_SAP_STATE		• Execution OK • Command not allowed for the current slave mode • Command not allowed for current SAP mode
Corresponding Output Macros:		
V1SL_C1_READ_DS()		

10.1.4.2 Signal the Completion of the Service ‘Write Data Set’ via the C1 Firmware

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c1_write_ds_done (void)
```

By calling this function, the user signals to the slave that a ‘Write Data Set’ service (*V1SL_C1_WRITE_DS()*) that was called previously is completed.

The reply data is transferred in the buffer used by the slave during the request. At the same time, write authorization for this buffer passes to the slave.

In the case of a positive acknowledgement, the reply data is to be structured according to a Write-RES-PDU, and in the case of a negative acknowledgement, according to a Write-NRS-PDU.

Note: If, prior to acknowledging the processing of the data set, the slave exits the DP mode *V1SL_DP_STATE_DATA_EX*, this function is to be called nevertheless. This is the precondition for restarting the C1 state machine, or for shutting down the slave (input function *v1sl_c0_withdraw()*).

This function can be called only after previously indicating the service ‘Write Data Set’ via the output macro *V1SL_C1_WRITE_DS()*, and only in the DP mode *V1SL_DP_STATE_DATA_EX*.

Input Function:		v1sl_c1_write_ds_done
Meaning:		Signal completion of service 'Write Data set' via the C1 firmware
Transfer:		
Parameter	Value Range	Meaning
Structure of Data Buffer:		
positive acknowledgement		_WRITE_DS_PTR→res (refer to Section 11.4.2 'C1/C2 DS_WRITE')
negative acknowledgement		_WRITE_DS_PTR→nrs (refer to Section 11.4.2 'C1/C2 DS_WRITE')
Return:		
Value Range		Meaning
V1SL_OK V1SL_ERR_SEQUENCE V1SL_ERR_SAP_STATE		• Execution OK • Command not allowed for the current slave mode • Command not allowed for current SAP mode
Corresponding Output Macros:		
V1SL_C1_WRITE_DS()		

10.1.5 Input Functions of C2 at the User Interface

10.1.5.1 Open a C2 Communication Channel

Prototype:

```
void V1SL_IFA_CODE_ATTR v1sl_c2_open_channel (SYS_PATH_TYPE sys_path)
```

By calling this function, the user opens a communication channel to the C2 firmware. This includes the allocation and initialization of resources needed by the C2 component of the V1SL. In addition, the communication channel to the layer below (PBC driver) is set up.

The user is informed of the completion of the function asynchronously via the output macro *V1SL_C2_OPEN_CHANNEL_DONE()*.

Note: Immediately after establishment of a communication channel, the connection is ready.

Calling this function is possible only after the V1SL has been initialized (input function *v1sl_init()*).

Input Function:			v1sl_c2_open_channel
Meaning:			Open a C2 communication channel
Transfer:			
Parameter	Value Range		Meaning
sys_path	(refer to parameter *V1SL_SYS_PATH_TYPE* in Section 25.1.		System path: ID of a C2 communication channel, assigned by the system; the system specifies the contrete type of this parameter.
Return:			
Value Range			Meaning
Corresponding Output Macros:			
V1SL_C2_OPEN_CHANNEL_DONE()			

10.1.5.2 Close a C2 Communication Channel

Prototype:

```
void V1SL_IFA_CODE_ATTR v1sl_c2_close_channel (void)
```

By calling this function, the user closes a communication channel to the C2 firmware. The resources allocated when the communication channel was opened are released. Also, the communication to the layer below (PBC driver) is closed.
The user is informed of the completion of the function asynchronously via the output macro *V1SL_C2_CLOSE_CHANNEL_DONE()*.

The function can be called only after the C2 communication channel has been opened (input function *v1sl_c2_open_channel()*).

Input Function:		**v1sl_c2_close_channel**
Meaning:		Close a C2 communication channel
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range		Meaning
Corresponding Output Macros:		
V1SL_C2_CLOSE_CHANNEL_DONE()		

10.1.5.3 Reply to an Initiate PDU

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c2_initiate_done (Unsigned8 con_id)
```

By calling this function, the user responds to a connection establishment request (output macro *V1SL_C2_INITIATE()*).

The response data is transferred in the buffer the slave used during the request. At the same time, write authorization for this data buffer passes to the slave.

If the user accepts the connection, it has to store an Initiate-RES-PDU in the data buffer.
The connection is rejected with an Abort-PDU.

Input Function:		**v1sl_c2_initiate_done**
Meaning:	Reply to an Initiate PDU	
Transfer:		
Parameter	Value Range	Meaning
con_id	000..connection_number-1 (refer to Section 11.3.1 'C2')	Reference of the connection whose establishment was requested
Structure of the Data Buffer:		
Connection accepted:	_INITIATE_PTR→res (refer to Section 11.3.3 'C2 INITIATE ')	
Connection rejected:	_INITIATE_PTR→nrs (refer to Section 11.3.3 'C2 INITIATE ')	
Connection rejected:	_INITIATE_PTR→abort (refer to Section 11.3.2 'C2 ABORT')	

Return:	
Value Range	Meaning
V1SL_OK	• Execution OK • In addition, it should be mentioned that wrong user actions cause the call of *V1SL_FATAL_ERROR()*, instead of returning a value with the cause for the error!
Corresponding Output Macros:	
V1SL_C2_INITIATE()	

10.1.5.4 Signal the Completion of the Service 'Data Transport'

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c2_data_transport_done (Unsigned8
con_id)
```

By calling this function, the user signals to the slave that a previously requested 'Data Transport' service (*V1SL_C2_DATA_TRANSPORT()*) has been completed.

The response data is transferred in the buffer the slave used during the request. At the same time, write authorization for this data buffer passes to the slave.
The response data is to be structured according to a Data_Transport-RES-PDU.

As an alternative, the user can initiate the shut down of the connection. In this case, an Abort-PDU is entered in the reply data buffer.

Input Function:		**v1sl_c2_data_transport_done**
Meaning:	Signal the completion of the service 'Data Transport'	
Transfer:		
Parameter	Value Range	Meaning
con_id	000..connection_number-1 (refer to Section 11.3.1 'C2' on page 107)	Reference of the connection used for the data transport
Structure of the Data Buffer:		
Reply Data:		_DATA_TRANSPORT_PTR→res (refer to Section 11.3.4)
Negative acknowledgement:		_DATA_TRANSPORT_PTR→nrs (refer to Section 11.3.4 C2 DATA_TRANSPORT')
Shut down of connection:		_DATA_TRANSPORT_PTR→abort (refer to Section 11.3.2 'C2 ABORT')
Return:		
Value Range		Meaning
V1SL_OK		• Execution OK • In addition, it should be mentioned that wrong user actions cause the call of *V1SL_FATAL_ERROR()*, instead of returning a value with the cause for the error!
Corresponding Output Macros:		
V1SL_C2_DATA_TRANSPORT()		

10.1.5.5 Signal the Completion of the Service 'Read Data Set' via the C2 Firmware

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c2_read_ds_done
(Unsigned8 con_id)
```

By calling this function, the user signals to the slave that the previously requested service 'Read Data Set' (*V1SL_C2_READ_DS()*)is completed.

The response data is transferred in the buffer the slave used during the request. At the same time, write authorization for this data buffer passes to the slave.

If the acknowledgement is positive, the response data is to be structured according to a Read-RES-PDU; if the acknowledgement is negative, according to a Read-NRS-PDU.

As an alternative, the user can initiate the shut down of the connection. In this case, an Abort-PDU is entered in the response data buffer.

Input Function:		**v1sl_c2_read_ds_done**
Meaning:	Signal the completion of the service 'Read Data Set' via the C2 Firmware	
Transfer:		
Parameter	Value Range	Meaning
con_id	000..connection_number-1 (refer to Section 11.3.1 'C2' on page 107)	Reference of the connection used for reading the data set.
Structure of the Data Buffer:		
positive acknowledgement		_READ_DS_PTR→res (refer to Section 11.4.1 'C1/C2 DS_READ')
negative acknowledgement		_READ_DS_PTR→nrs (refer to Section 11.4.1 'C1/C2 DS_READ'))
shut down of connection		_READ_DS_PTR→abort (refer to Section 11.4.2 'C2 ABORT')
Return:		
Value Range		Meaning
V1SL_OK		• Execution OK • In addition, it should be mentioned that wrong user actions cause the call of *V1SL_FATAL_ERROR()*, instead of returning a value with the cause for the error!
Corresponding Output Macros:		
V1SL_C2_READ_DS()		

10.1.5.6 Signal the Completion of the Service 'Write Data Set' via the C2 Firmware

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c2_write_ds_done (Unsigned8 con_id)
```

By calling this function, the user signals to the slave that the previously requested service 'Write Data Set' (*V1SL_C2_WRITE_DS()*) is completed.

The response data is transferred in the buffer the slave used during the request. At the same time, write authorization for this data buffer passes to the slave.

If the acknowledgement is positive, the response data is to be structured according to a Write-RES-PDU; if the acknowledgement is negative, according to a Write-NRS-PDU.

As an alternative, the user can initiate the shut down of the connection. In this case, an Abort-PDU is entered in the response data buffer.

Input Function:		**v1sl_c2_write_ds_done**
Meaning:	Signal the completion of the service 'Write Data Set' via the C2 firmware	
Transfer:		
Parameter	Value Range	Meaning
con_id	000..connection_number-1 (refer to Section 11.3.1 'C2')	Reference of the connection used for writing the data set.
Structure of the Data Buffer:		
positive acknowledgement		_WRITE_DS_PTR→res (refer to Section 11.4.2 'C1/C2 DS_WRITE')
negative acknowledgement		_WRITE_DS_PTR→nrs (refer to Section 11.4.2 'C1/C2 DS_WRITE')
shut down of connection		_WRITE_DS_PTR→abort (refer to Section 11.3.2 'C2 ABORT')
Return:		
Value Range		Meaning
V1SL_OK		• Execution OK • In addition, it should be mentioned that wrong user actions cause the call of *V1SL_FATAL_ERROR()*, instead of returning a value with the cause for the error!
Corresponding Output Macros:		
V1SL_C2_WRITE_DS()		

10.1.5.7 User Cancels a Connection

Prototype:

```
Unsigned8 V1SL_IFA_CODE_ATTR v1sl_c2_user_abort (Unsigned8 con_id,
Unsigned8 subnet, Unsigned8 reason_code)
```

By calling this function, the user can initiate the shut down of a connection, regardless of whether a request PDU that needs to be answered is present at the user.

After accepting this function call, the DPV1 slave responds to the next message that is received with an Abort-PDU.

The DPV1 slave acknowledges asynchronously with *V1SL_C2_USER_ABORT_DONE()* after the connection was cancelled.

This function can be called only after a connection was established successfully (*V1SL_C2_INITIATE()*).

Input Function:		**V1sl_c2_user_abort**
Meaning:	User cancels a connection	
Transfer:		
Parameter	Value Range	Meaning
con_id	000..connection_number-1 (refer to Section 11.3.1 'C2')	Reference of the connection to be canceled
subnet	(refer to Section 12.3.3 'Parameter subnet at an Abort -PDU')	Identifies location of the connection shut down
reason_code	(refer to Section 0 ' Parameter reason_code of an Abort-PDU' on page 124)	Triggering firmware component and cause for connection shut down
Return:		
Value Range	Meaning	
V1SL_OK	• Execution OK • In addition, it should be mentioned that wrong user actions cause the call of *V1SL_FATAL_ERROR()*, instead of returning a value with the cause for the error!	
Corresponding Output Macros:		
V1SL_C2_USER_ABORT_DONE()		

10.1.6 Interface Expansion of the Input Functions in Multi-Instance Operation

10.1.6.1 General

All input functions of the C0 and C2 firmware receive, when multi-instance operation is activated, an additional parameter that is used for referencing the communication channel. Two types of input functions are to be differentiated based on the deviating meaning of this additional parameter (*handle*).

The presentation of the function prototypes used below explains only the differences in the call parameters in comparison to the non-multi-instance variant of the respective function. For that reason, the prototype presentation is interrupted with '…', and not specified completely.

10.1.6.2 Open a Communication Channel

Prototype:

```
void V1SL_IFA_CODE_ATTR v1sl_c0_open_channel (..., Unsigned8 handle)
```

```
void V1SL_IFA_CODE_ATTR v1sl_c2_open_channel (..., Unsigned8 handle)
```

Input Function:		**v1sl_c0_open_channel/v1sl_c2_open_channel**
Meaning:		Open a communication channel
Transfer:		
Parameter	Value Range	Meaning
...		Parameters that depend on the concrete function
Handle		User handle; during further calls of output macros via this communication channel, V1SL copies this value to the element *_HANDLE* (refer to Section 0 'Interface Expansion of the Output Macros in Multi-Instance Operation').
Return:		
Value Range		Meaning
Corresponding Output Macros:		
V1SL_C0_OPEN_CHANNEL_DONE(), V1SL_C2_OPEN_CHANNEL_DONE()		

10.1.6.3 Close a Communication Channel, Request Functions

Prototype:

```
... V1SL_IFA_CODE_ATTR v1sl_c0_... (..., Unsigned8 handle)

... V1SL_IFA_CODE_ATTR v1sl_al... (..., Unsigned8 handle)

... V1SL_IFA_CODE_ATTR v1sl_c1_... (..., Unsigned8 handle)

... V1SL_IFA_CODE_ATTR v1sl_c2_... (..., Unsigned8 handle)
```

Input Function:		**v1sl_c0_.../v1sl_al_.../v1sl_c1_.../v1sl_c2_...**
Meaning:		Close a communication channel, request functions
Transfer:		
Parameter	Value Range	Meaning
...		Parameters that depend on the concrete function
handle	000..254 V1SL_ HANDLE_ EMPTY	V1SL handle; the user received this value with the acknowledgement of the open channel request *V1SL_C0_OPEN_CHANNEL_DONE()*, or *V1SL_C2_OPEN_CHANNEL_DONE()* for referencing the communication channel within V1SL: • Area used by V1SL • Impermissible value
Return:		
Value Range		Meaning
...		Depends on the concrete function
Corresponding Output Macros:		
V1SL_C0_...(), V1SL_AL_...(), V1SL_C1_...(), V1SL_H_...(), V1SL_SC_...(), V1SL_S7_...(), V1SL_C2_...()		

10.2 Output Macros

Note: Output macros can also be called within the context (during processing) of input functions!

10.2.1 Overview

C0-Output Macros	Description
V1SL_C0_OPEN_CHANNEL_DONE	Acknowledges establishment of a C0 communication channel
V1SL_C0_CLOSE_CHANNEL_DONE	Acknowledges shut down of a C0 communication channel
V1SL_C0_WITHDRAW_DONE	Acknowledges release of a slave's memory resources
V1SL_C0_DP_WD_TIMEOUT	Indicates expiration of DP watchdog
V1SL_C0_WD_STATE_REPORT	Displays DP watchdog state change

V1SL_C0_DP_STATE_REPORT	Displays DP state change
V1SL_C0_LED_STATE_REPORT	Displays recommended bus error LED state
V1SL_C0_DATA_EXCHANGE_ACTIVE	Displays start of user data exchange
V1SL_C0_REAL_CFG_BUFFER_CHANGED	Indicates availability of a new expected configuration data buffer
V1SL_C0_NEW_SSA	Indicates new ‘Set Slave Address‘ telegram
V1SL_C0_NEW_PRM	Indicates new parameterization data
V1SL_C0_NEW_CFG	Indicates new configuration data sent by the master
V1SL_C0_CLEAR	Indicates CLEAR/UNCLEAR
V1SL_C0_SYNC	Indicates new GC command SYNC/UNSYNC
V1SL_C0_FREEZE	Indicates new GC command FREEZE/UNFREEZE
V1SL_C0_DIAG_CHANGED	Returns user’s diagnostic data buffer
V1SL_C0_DIAG_FETCHED	Message ‘Parameterization master fetched diagnostic‘
AL-Output Macros	**Description**
V1SL_AL_STATE_REPORT	Indicates state change of the alarm state machine
V1SL_AL_ALARM_ACK	Acknowledges alarm
C1-Output Macros	**Description**
V1SL_C1_READ_DS	Indicates request ‘Read Data Set‘ via C1 firmware
V1SL_C1_WRITE_DS	Indicates request ‘Write Data Set‘ via C1 firmware
C2-Output Macros	**Description**
V1SL_C2_OPEN_CHANNEL_DONE	Acknowledges establishment of a C2 communication channel
V1SL_C2_CLOSE_CHANNEL_DONE	Acknowledges shut down of a C2 communication channel
V1SL_C2_INITIATE	Indicates request 'INITIATE'
V1SL_C2_ABORT	Indicates request 'ABORT'
V1SL_C2_USER_ABORT_DONE	Acknowledges request 'USER ABORT'
V1SL_C2_DATA_TRANSPORT	Indicates request 'DATA_TRANSPORT'
V1SL_C2_READ_DS	Indicates request ‘Read Data Set‘ via C2 firmware
V1SL_C2_WRITE_DS	Indicates request ‘Write Data Set‘ via C2 firmware

10.2.2 Output Macros of C0 at User Interface

10.2.2.1 Acknowledge Establishment of a C0 Communication Channel

Prototype:

```
#define V1SL_C0_OPEN_CHANNEL_DONE(_RETURN_VALUE)
```

By calling this macro, V1SL acknowledges the input function *v1sl_c0_open_channel()* called by the user. This acknowledgement is made after the communication channel was opened.
The V1SL now expects as the next action that the user sets up the memory resources (input function *v1sl_c0_add()*).

Output Macro:		**V1SL_C0_OPEN_CHANNEL_DONE**
Meaning:	Acknowledge establishment of a C0 communication channel	
Transfer:		
Parameter	Type, Attribute / Value Range	Meaning
_RETURN_VALUE	Unsigned8 / V1SL_OK V1SL_ERR_SEQUENCE V1SL_ERR_LOWER_LAYER V1SL_ERR_HANDLE V1SL_ERR_INT_DATA V1SL_ERR_PATH	Display of result: • Execution OK • Command not allowed for the current slave mode • Establishment of the communication channel to PBC driver was acknowledged negative • No free handle/communication channel available • Allocation of internal (*V1SL_INT_DATA_ATTR*) data memory was acknowledged negative • Call of *V1SL_C0C2_GET_PATH_INFO()* was acknowledged negative
Recommended Actions:		
• Set up memory resources by calling *v1sl_c0_add()*		
Corresponding Input Functions:		
v1sl_c0_open_channel()		

10.2.2.2 Acknowlede Shut Down of a C0 Communication Channel

Prototype:

```
#define V1SL_C0_CLOSE_CHANNEL_DONE(_RETURN_VALUE)
```

By calling this macro, V1SL acknowledges the input function *v1sl_c0_close_channel()* called by the user. This acknowledgement is made after the C0 communication has been closed.

Output Macro:		**V1SL_C0_CLOSE_CHANNEL_DONE**
Meaning:	Acknowledge shut down of a C0 communication channel	
Transfer:		
Parameter	Type, Attribute / Value Range	Meaning
_RETURN_VALUE	Unsigned8 / V1SL_OK V1SL_ERR_SEQUENCE V1SL_ERR_LOWER_LAYER	Display of result • Execution OK • Command not allowed for the current slave mode • Shut down of the communication channel to the PBC driver was acknowledged negative
Recommended Actions:		
• Reestablishment of communication channel is possible with *v1sl_c0_open_channel()*		
Corresponding Input Functions:		
v1sl_c0_close_channel()		

10.2.2.3 Acknowledge Release of the Slave's Memory resources

Prototype:

```
#define V1SL_C0_WITHDRAW_DONE()
```

By calling this macro, V1SL acknowledges the input function *v1sl_c0_withdraw()* called by the user. This acknowledgement is made after the slave's memory resources were released.

Output Macro:	V1SL_C0_WITHDRAW_DONE	
Meaning:	Acknowldge release of the slave's memory resources	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
Recommended Actions:		
• Set up new slave memory resources is possible by calling *v1sl_c0_add()* • Shut down the C0 communication channel is possible by calling *v1sl_c0_close_channel()*		
Corresponding Input Functions:		
v1sl_c0_withdraw()		

10.2.2.4 Indicate Timeout of DP Watchdog

Prototype:

```
#define V1SL_C0_DP_WD_TIMEOUT()
```

By calling this macro, the slave indicates that the DP watchdog has expired. The reasons for the timeout are not part of the information.
The slave can call the output macro only if the slave instance has been started (*v1sl_c0_control()*).

Output Macro:	V1SL_C0_DP_WD_TIMEOUT	
Meaning:	Indicates timeout of DP watchdog	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
Recommended Actions:		
•		
Corresponding Input Functions:		

10.2.2.5 Displaying a DP Watchdog State Change

Prototype:

```
#define V1SL_C0_WD_STATE_REPORT(_STATE)
```

By calling this macro, the V1SL indicates to the user changes in the DP-WD state.
This macro is called only if the slave's memory resources are set up (*v1sl_c0_add()*).

Output Macro:	V1SL_C0_WD_STATE_REPORT	
Meaning:	Indicates a DP watchdog state change	
Transfer:		
Parameter	Type, Attribute / Value Range	Meaning
_STATE	Unsigned8 / (refer to Section 12.2.4 'DP Watchdog States' on page 119) V1SL_WD_STATE_ BAUD_SEARCH	Current DP slave watchdog state: • Watchdog is in mode baudrate search

	V1SL_WD_STATE_ BAUD_CONTROL V1SL_WD_STATE_ DP_MODE	• Watchdog is in mode monitoring baudrates • Watchdog is in the DP mode; that is, the DP data traffic is being monitored; refer to data *wd_fact_1* and *wd_fact_2* in Table 10
Recommended Actions:		
• Trigger user state machine with the slave state		
Corresponding Input Functions:		

10.2.2.6 Display of a DP Mode Change

Prototype:

```
#define V1SL_C0_DP_STATE_REPORT(_STATE)
```

By calling this macro, the V1SL indicates to the user that the DP slave state has changed. The information about the slave states is as follows:

- *V1SL_DP_STATE_OFF*: The slave is deactivated, and can not be parameterized by any master, or otherwise addressed on the bus.
- *V1SL_DP_STATE_NO_DATA_EX*: The slave is activated, and can be parameterized by a master; there is no cyclical user data exchange. The output macro can be called several time in succession with this parameter, which means a release <<reset?>> of the internal state machines.
- *V1SL_DP_STATE_DATA_EX*: The parameterization master successfully parameterized and configured the slave, and the slave is ready for cyclical data exchange. At this time, user data transfer is not yet mandatory (refer to Section 6.5) The output macro is called only if the slave's memory resources are set up (*v1sl_c0_add()*).

Output Macro:	**V1SL_C0_DP_STATE_REPORT**	
Meaning:	Indicate a DP mode change	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_STATE	Unsigned8 / (refer to Section 12.2.5 'DP' on page 119) V1SL_DP_STATE_ INVALID V1SL_DP_STATE_ OFF V1SL_DP_STATE_ NO_DATA_EX V1SL_DP_STATE_ DATA_EX	Current DP slave state: • Initial mode; not signalled to the user! • Slave is not active on the bus-side, and responds to each master access with 'service not activated' (RS). • Slave is not in the cyclical data exchange mode • Slave is in the cyclical data exchange mode with the parameterization master
Recommended Actions:		
•		
Corresponding Input Functions:		

10.2.2.7 Display a Recommended Bus Error LED State

Prototype:

```
#define V1SL_C0_LED_STATE_REPORT(_STATE)
```

By calling this macro, the V1SL indicates to the user a bus error LED state. The coding used for this is very common and can be found in many DP slave modules.
The states of the bus error LED are assigned to the following situations in the slave:

Baud Control Timer Mode: V1SL_WD_STATE_BAUD_SEARCH	**Slave State:** V1SL_DP_STATE_DATA_EX	**Resulting Bus Error LED State:** off	on	blinking
no	no			X
no	yes	X		
yes	yes	X		
yes	no		X	

The blinking frequency of the bus error LED is determined by the system environment; a frequency of 0.7 Hz is recommended.
The output macro is called only if the slave's memory resources are set up (*v1sl_c0_add()*).

Output Macro:		**V1SL_C0_LED_STATE_REPORT**
Meaning:	Display a recommended bus error LED state	
Transfer:		
Parameter	Type, Attribute / Value Range	Meaning
_STATE	Unsigned8 / (refer to Section 12.2.4 'Bus Error LED States') V1SL_LED_STATE_OFF V1SL_LED_STATE_ON V1SL_LED_STATE_FLASH	Current bus error LED state: • Switch bus error LED off • Switch bus error LED on • Bus error LED is blinking with a frequency specified by the system
Recommended Actions:		
• Update the display of the bus error LED		
Corresponding Input Functions:		

10.2.2.8 Indicate Start of User Data Exchange

Prototype:

```
#define V1SL_C0_DATA_EXCHANGE_ACTIVE()
```

By calling this macro, the V1SL indicates the start of user data exchange between parameterization master and slave to the user. Necessary requirements for this indication are:

- The slave is in the DP mode *V1SL_DP_STATE_DATA_EX*
- The user has transmitted the message 'Application_Ready' (*v1sl_c0_control()*) to the V1SL
- At least one user data telegram was received from the parameterization master.

The output macro is called only if the slave's memory resources are set up (*v1sl_c0_add()*).

Output Macro:		**V1SL_C0_DATA_EXCHANGE_ACTIVE**
Meaning:	Indicates start of user data exchange	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
Recommended Actions:		
•		
Corresponding Input Functions:		

10.2.2.9 Indicate the Availability of a New Buffer with Expected Configuration Data

Prototype:

```
#define V1SL_C0_REAL_CFG_BUFFER_CHANGED()
```

By calling this macro, the slave signals to the user that a new buffer for expected configuration data is available. The user can fetch this buffer from the slave by calling the input function *v1sl_c0_get_real_cfg_ptr()* and enter its expected configuration data. After that, updated expected configuration data can be made available to the slave by calling the input function *v1sl_c0_real_cfg_update()*.
The slave calls the output macro for the first time when setting up the memory resources (*v1sl_c0_add()*), and after each update of the expected configuration by the user (input function *v1sl_c0_real_cfg_update()*).

Output Macro:		**V1SL_C0_REAL_CFG_BUFFER_CHANGED**
Meaning:	Indicates availability of a new expected configuration data buffer	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
Recommended Actions:		
• Fetch new expected configuration data buffer with *v1sl_c0_get_real_cfg_ptr()*, • Enter expected configuration data in fetched buffer, • Update expected cofiguration data with *v1sl_c0_real_cfg_update()*.		
Corresponding Input Functions:		
v1sl_c0_get_real_cfg_ptr(), v1sl_c0_real_cfg_update()		

10.2.2.10 Indicate New 'Set Slave Address' Telegram

Prototype:

```
#define V1SL_C0_NEW_SSA(_SSA_PTR,_SSA_LEN)
```

By calling this macro, the V1SL indicates to the user the receipt of a new 'Set Slave Address' telegram. At the time of the indication, the slave has already accepted the station address specified in the telegram (element *slave_address*). This applies also to

the information whether additional changes of the station address are permitted (element *no_address_change*).
Before the user receives the indication of new slave address data, the V1SL stops the slave on its own intiative (*V1SL_C0_DP_STATE_REPORT()*). The user thus has a neutral start position in order to decide on the further firmware processing:

- Restart slave with new station address; to do this, the user has to carry out the following sequence:
  - ♦ Acknowlede processing of the 'Set Slave Address' telegram with *v1sl_c0_control()*,
  - ♦ Restart the slave with *v1sl_c0_control()*.
- Shut down the slave, and preparation of startup with a different set of slave memory resources; to do this, the user has to carry out the following sequence:
  - ♦ Acknowledge processing of the 'Set Slave Address' telegram with *v1sl_c0_control()*,
  - ♦ Enable the slave's memory resources with *v1sl_c0_withdraw()*.

The slave can call the output macro only if the slave instance is started (*v1sl_c0_control()*).

Output Macro:		**V1SL_C0_NEW_SSA**
Meaning:	Indicate new 'Set Slave Address' telegram	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_SSA_PTR	V1SL_LL_SSA_PTR	Pointer to 'Set Slave Address' data
_SSA_LEN	Unsigned8 / 000..c0_ssa_buffer_len (refer to Section 11.2.2 'C0')	Length of the received 'Set Slave Address' telegram
Recommended Actions:		
• The user has to acknowledge with *v1sl_c0_control()* in any case		
Corresponding Input Functions:		
v1sl_c0_control()		

10.2.2.11 Indicating New Parameterization Data

Prototype:

```
#define V1SL_C0_NEW_PRM(_PRM_PTR,_PRM_LEN)
```

By calling this macro, the slave transfers new parameterization data to the user.
Before the user is triggered with this output macro, the slave validated the data relevant to DPV1. The completion of processing the data has to be acknowledged to the slave by calling the input function *v1sl_c0_control()*.

Note: If, prior to the user setting the result of the parameterization data check, the output macro is called again, previous processing is to be cancelled immediately and the checked data are no longer valid. The newly received data is to be checked.

The slave can call the output macro only if the slave instance was started (*v1sl_c0_control()*).

Output Macro:	V1SL_C0_NEW_PRM	
Meaning:	Indicate new parameterization data	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_PRM_PTR	V1SL_LL_PRM_PTR	Pointer to parameterization data
_PRM_LEN	Unsigned8 / 007..c0_prm_buffer_len (refer to Section 11.2.2'C0')	Length of the parameterization data telegram
Recommended Actions:		
• Evaluate data • Transmit *v1sl_c0_control(V1SL_CONTROL_PRM_OK)* acknowledgment if OK • Transmit *v1sl_c0_control(V1SL_CONTROL_PRM_ERROR)* acknowledgement if an error occurs		
Corresponding Input Functions:		
v1sl_c0_control()		

10.2.2.12 Indicate New Configuration data sent by the master

Prototype:

```
#define V1SL_C0_NEW_CFG(_CFG_PTR,_CFG_LEN,_MODE)
```

By calling this macro, the slave transfers new configuration data sent by the master to the user for comparison with the expected configuration. The completion has to be acknowledged to the slave by calling the input function *v1sl_c0_control()*.

Note: If, prior to the user setting the result of the configuration data check, the output macro is called again, previous processing is to be cancelled immediately and the checked data are no longer valid. The newly received data is to be checked.

The slave can call the output macro only if the slave instance is started (*v1sl_c0_control()*).

Output Macro:	V1SL_C0_NEW_CFG	
Meaning:	Transfer new configuration sent by the master to the user	
Transfer:		
Parameter	Type, Attribute / Value Range	Meaning
_CFG_PTR	V1SL_LL_UNSIGNED8_PTR	Pointer to configuration data sent by the master
_CFG_LEN	Unsigned8/ 000..c0_cfg_buffer_len (refer to Section 11.2.2 'C0')	Length of configuration data sent by the master
_MODE	Unsigned8 / (refer to Section 12.2.6 'Configuration Parameter') V1SL_CFG_MODE_ STOP_ON_FAULT V1SL_CFG_MODE_ RUN_ON_FAULT	Configuration check requirement: • Configuration sent by the master and expected configuraton have to agree 100% for startup to be possible • Deviations at the configuration comparison are tolerated; startup possible
Recommended Actions:		
• Compare configuration sent by master and expected configuration • Take *_MODE* into consideration when specifying the result of the comparison • Transmit *v1sl_c0_control(V1SL_CONTROL_CFG_OK or V1SL_CONTROL_CFG_UPDATE)* request if		

OK • Transmit *v1sl_c0_control(V1SL_CONTROL_CFG_ERROR)* request if an error occurs
Corresponding Input Functions:
v1sl_c0_control()

10.2.2.13 Indicate CLEAR/UNCLEAR

Prototype:

```
#define V1SL_C0_CLEAR(_STATE)
```

By calling this macro, the slave indicates to the user a state transition from CLEAR→UNCLEAR and vice versa. The call with the 'CLEAR' parameter can be made multiple times.

V1SL_C0_CLEAR(TRUE) is indicated under the following conditions:

- Global Control Command 'CLEAR' from parameterization master was received
- Parameterization master enters 'Failsafe' mode (transmission of user data telegrams with output data length = 0 in the case of existing outputs)
- Immediately after leaving data exchange mode (slave's DP mode *V1SL_DP_STATE_NO_DATA_EX*).

V1SL_C0_CLEAR(FALSE) is indicated under the following condition:

- Global Control Command 'UNCLEAR' was received.

The slave can call the output macro only if the slave instance is started (*v1sl_c0_control()*).

Output Macro:		**V1SL_C0_CLEAR**
Meaning:	Indicate CLEAR/UNCLEAR	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_STATE	Unsigned8 / TRUE FALSE	Indication: • Slave has changed to the 'Clear' mode • Slave has changed to the 'Unclear' mode (also called 'Operate')
Recommended Actions:		
• User-specific		
Corresponding Input Functions:		

10.2.2.14 Indicating New GC Command SYNC/UNSYNC

Prototype:

```
#define V1SL_C0_SYNC(_STATE)
```

By calling this macro, the slave indicates to the user a control command SYNC or UNSYNC from the parameterization master. The call can be made with the same parameter multiple times. In most cases, user response is not required.

Note: If needed, the 'SYNC' functionality can be activated during the slave's power-up.

Note: Since the slave does not acknowledge the Global Control Commands to the parameterization master, they may be missed. The user can not rely on receiving them, and therefore can not count on receiving an indication via the output macro.

The slave can call the output macro only if the slave instance is started (*v1sl_c0_control()*).

Output Macro:		**V1SL_C0_SYNC**
Meaning:	Indicate new GC command SYNC/UNSYNC	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_STATE	Unsigned8 / TRUE FALSE	Indication: • Slave has received a SYNC command • Slave has received an UNSYNC command
Recommended Actions:		
• User-specific		
Corresponding Input Functions:		

10.2.2.15 Indicate New GC Command FREEZE/UNFREEZE

Prototype:

```
#define V1SL_C0_FREEZE(_STATE)
```

By calling this macro, the slave indicates to the user a control command FREEZE or UNFREEZE from the parameterization master. The call can be made multiple times with the same parameter. In most cases, the user does not have to respond.

Note: If needed, the 'FREEZE functionality can be activated during the slave's power-up.

Note: Since the slave does not acknowledge the Global Control Commands to the parameterization master, they may be missed. The user can not rely on receiving them, and therefore can not count on receiving an indication via the output macro.

The slave can call the output macro only if the slave instance is started (*v1sl_c0_control()*).

Output Macro:		**V1SL_C0_FREEZE**
Meaning:	Indicate new GC command FREEZE/UNFREEZE	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_STATE	Unsigned8 / TRUE FALSE	Indication: • Slave received a FREEZE command • Slave received an UNFREEZE command
Recommended Actions:		
• User-specific		

Corresponding Input Functions:

10.2.2.16 Return User's Diagnostic Data Buffer

Prototype:

```
#define V1SL_C0_DIAG_CHANGED(_USER_DIAG)
```

By calling this macro, the slave returns a diagnostic buffer to the user that was previously transferred with the input function *v1sl_c0_set_diag()*.

Note: Calling this output macro provides no information about whether the content of the returned diagnostic buffer was fetched by the parameterization master. For this, the output macro. *V1SL_C0_DIAG_FETCHED()* is used.

The slave can call the output macro only after the user set a diagnostic (input function *v1sl_c0_set_diag()*).

Output Macro:		**V1SL_C0_DIAG_CHANGED**
Meaning:	Return user's diagnostic data buffer	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_USER_DIAG	V1SL_UNION_DIAG_PTR	User buffer with diagnostic data previously transferred to the V1SL via the service *v1sl_c0_set_diag()*.
Recommended Actions:		
• Mark diagnostic data buffer in the user as 'free' • Enter new diagnostic data if necessary • Update diagnostic with *v1sl_c0_set_diag()*.		
Corresponding Input Functions:		
v1sl_c0_set_diag()		

10.2.2.17 Indication 'Parameterization Master Fetched Diagnostic'

Prototype:

```
#define V1SL_C0_DIAG_FETCHED(_USER_ID)
```

By calling this macro, the slave indicates to the user that the parameterization master fetched a previously set diagnostic information. As transfer parameter, the user receives the user identification of its diagnostic buffer which was transferred to the slave with *v1sl_c0_set_diag()*.

The slave calls the output macro only after the user set a diagnostic (via input function *v1sl_c0_set_diag()*), and only if the slave exchanges user data with the parameterization master (*V1SL_C0_DATA_EXCHANGE_ACTIVE()*).

Output Macro:	V1SL_C0_DIAG_FETCHED	
Meaning:	Indication 'Parameterization master fetched diagnostic'	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_USER_ID	V1SL_C0_USER_ID_TYPE	Identification of the user's diagnostic buffer that the parameterization master fetched, and which was previously transferred to the V1SL via the service *v1sl_c0_set_diag()*.
Recommended Actions:		
• Evaluate *_USER_ID* and act accordingly • Search for diagnostic data buffer within the application that is marked 'free' • Enter new diagnostic data if necessary • Update diagnostic with *v1sl_c0_set_diag()*.		
Corresponding Input Functions:		
v1sl_c0_set_diag()		

10.2.3 Output Macros of AL at the User Interface

10.2.3.1 Indicate State Change of the Alarm State Machine

Prototype:

```
#define V1SL_AL_STATE_REPORT(_ALARM_TYPE_BIT_FIELD,
_SEQUENCE_DEPTH)
```

By callling this macro, the slave indicates to the user the activation or deactivation of the alarm state machine, specifying the permissible alarm types and the supported alarm queue.
The slave can call the output macro only if the slave instance is started (*v1sl_c0_control()*).

Output Macro:	V1SL_AL_STATE_REPORT	
Meaning:	Indicate a state change of the alarm state machine	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_ALARM_TYPE_BIT_FIELD	Unsigned8 / (refer to Section 12.2.10 'Alarm') V1SL_ALARM_TYPE_ NONE_VALUE Otherwise, the V1SL supplies a bit field consisting of the OR operation of the following values V1SL_ALARM_TYPE_ DIAG_VALUE V1SL_ALARM_TYPE_ PROC_VALUE V1SL_ALARM_TYPE_	With this parameter V1SL indicates the permissible alarm types: • The alarm state machine was disabled; set alarms is no longer allowed for the user. The user **must** withdraw alarms already set by calling the input function *v1sl_al_withdraw_alarm()*. If this is not done, the alarm state machine will no longer be activated after the slave reenters the DP mode *V1SL_DP_STATE_DATA_EX* ! • The alarm state machine was activated.

	PUPL_VALUE V1SL_ALARM_TYPE_ STAT_VALUE V1SL_ALARM_TYPE_ UPDT_VALUE V1SL_ALARM_TYPE_ MANU_VALUE	
_SEQUENCE_DEPTH	Unsigned8 / (refer to Section 12.2.10 'Alarm') V1SL_SEQC_MODE_ TOTAL_00 V1SL_SEQC_MODE_ OFF V1SL_SEQC_MODE_ TOTAL_02 V1SL_SEQC_MODE_ TOTAL_04 V1SL_SEQC_MODE_ TOTAL_08 V1SL_SEQC_MODE_ TOTAL_12 V1SL_SEQC_MODE_ TOTAL_16 V1SL_SEQC_MODE_ TOTAL_24 V1SL_SEQC_MODE_ TOTAL_32	Alarm mode/ number of alarms (for information only; no conclusions regarding the alarm behavior of the user have to be made): • Alarm state machine deactivated; no alarms are to be sent or set by the user • Alarm state machine is not processing in the sequence mode but in the type mode; that is, one alarm each of each type is permitted to be active on the bus at one point in time • Sequence mode; 2 alarms of any type may be processed at one point in time with the parameterization master • Sequence mode; 4 alarms... • Sequence mode; 8 alarms... • Sequence mode; 12 alarms... • Sequence mode; 16 alarms... • Sequence mode; 24 alarms... • Sequence mode; 32 alarms...
Recommended Actions:		
• If needed, set the permissible alarms to the DPV1 slave by calling of *v1sl_al_set_alarm()*. • If needed, withdraw alarms by calling *v1sl_al_withdraw_alarm()*.		
Corresponding Input Functions:		
v1sl_al_set_alarm(), v1sl_al_withdraw_alarm()		

10.2.3.2 Acknowledge Alarm

Prototype:

```
#define V1SL_AL_ALARM_ACK(_ALARM_PTR)
```

By calling this macro, the slave acknowledges an alarm to the user that was set previously:

- The slave receives the acknowledgement in DPV1 operation from the parameterization master, and tranfers it to the user.
- In addition, the output macro is called if the user withdraws alarms by calling *v1sl_al_withdraw_alarm()*.

The slave can call the output macro only after an alarm was set by the user (via input function *v1sl_al_set_alarm()*).

Output Macro:	V1SL_AL_ALARM_ACK	
Meaning:	Acknowlede alarm	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_ALARM_PTR	V1SL_IFA_ALARM_PTR	Buffer with alarm data transferred to V1SL with the service *v1sl_al_set_alarm()* by the user.
Recommended Actions:		
• Acknowledge the pending alarm to the periphery modules • In the case of intelligent slaves, complete the system call (SFC) that triggered the alarm.		
Corresponding Input Functions:		
v1sl_al_set_alarm()		

10.2.4 Output Macros of C1 at the User Interface

10.2.4.1 Indicate the Request 'Read Data Set' via the C1 Firmware

Prototype:

```
#define V1SL_C1_READ_DS(_READ_DS_PTR)
```

By calling this macro, the slave transfers the request for reading a data set to the user. The user has to acknowledge the completion of reading the data set by calling the function *v1sl_c1_read_ds_done()*.
The slave uses the output macro only after the C0 communication channel has been started (*v1sl_c0_control()*).

Output Macro:	V1SL_C1_READ_DS	
Meaning:	Indicate the request 'Read Data Set' via the C1 firmware	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_READ_DS_PTR	V1SL_LL_DS_READ_PTR	Pointer to the Read-REQ-PDU: • _READ_DS_PTR→req (refer to Section 11.4.1 'C1/C2 DS_READ')
Recommended Actions:		
• Evaluate transfer parameters • Read the specified data set, and enter it in the specified PDU • Acknowledge the request by calling *v1sl_c1_read_ds_done()*		
Corresponding Input Functions:		
v1sl_c1_read_ds_done()		

10.2.4.2 Indicate the Request 'Write Data Set' via the C1 Firmware

Prototype:

```
#define V1SL_C1_WRITE_DS(_WRITE_DS_PTR)
```

By calling this macro, the slave transfers to the user a request for writing a data set. The user has to acknowledge the completion of writing the data set by calling the function *v1sl_c1_write_ds_done()*.

The slave uses the output macro only after the C0 communication channel was started (*v1sl_c0_control()*).

Output Macro:		**V1SL_C1_WRITE_DS**
Meaning:	Indicate the request 'Write Data Set' via the C1 firmware	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_WRITE_DS_PTR	V1SL_LL_DS_WRITE_PTR	Pointer to the Write-REQ-PDU: • _WRITE_DS_PTR→req (refer to Section 11.4.2 'C1/C2 DS_WRITE')
Recommended Actions:		
• Evaluate the transfer parameters • Write the transferred data set, or route it to the destination component • Acknowledge the request by calling *v1sl_c1_write_ds_done()*		
Corresponding Input Functions:		
v1sl_c1_write_ds_done()		

10.2.5 Output Macros of C2 at the User Interface

10.2.5.1 Acknowledge Establishment of a C2 Communication Channel

Prototype:

```
#define V1SL_C2_OPEN_CHANNEL_DONE(_RETURN_VALUE)
```

V1SL acknowledges the input function *v1sl_c2_open_channel()* called by the user. This acknowledgement is made after the communication channel was opened. This establishes the connection readiness via the C2 communication channel. It is possible to receive connection establishment requests (*V1SL_C2_INITIATE()*).

Output Macro:		**V1SL_C2_OPEN_CHANNEL_DONE**
Meaning:	Acknowledge establishment of a C2 communication channel	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_RETURN_VALUE	Unsigned8 / V1SL_OK V1SL_ERR_SEQUENCE V1SL_ERR_PATH V1SL_ERR_LOWER_LAYER V1SL_ERR_TIMER V1SL_ERR_SAP V1SL_ERR_POLL_TIMEOUT	Result indication: • Execution OK • Request not permittted in present mode • Negative acknowledgement for request *V1SL_C0C2_GET_PATH_INFO()* • Establish the communication channel to the layer below was acknowledged negatively • No timers can be allocated • Impermissible SAP number declared via the detail pointer • Impermissible monitoring time declared via the detail pointer
Recommended Actions:		
• Wait for connection establishment request (output macro *V1SL_C2_INITIATE()*)		
Corresponding Input Functions:		
v1sl_c2_open_channel()		

10.2.5.2 Acknowledge the Shut Down of a C2 Communication Channel

Prototype:

```
#define V1SL_C2_CLOSE_CHANNEL_DONE(_RETURN_VALUE)
```

V1SL acknowledges the input function *v1sl_c2_close_channel()* called by the user. This acknowledgement is made after the C2 communication channel was closed. This also cancels the connection readiness via the C2 communication channel.

Output Macro:	**V1SL_C2_CLOSE_CHANNEL_DONE**	
Meaning:	Acknowledge the shut down of a C2 communication channel	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_RETURN_VALUE	Unsigned8 / V1SL_OK V1SL_ERR_SEQUENCE	Result indication • Execution OK • Request not permitted in the present mode
Recommended Actions:		
• Reestablishment of the communication channel is possible with *v1sl_c2_open_channel()*		
Corresponding Input Functions:		
v1sl_c2_close_channel()		

10.2.5.3 Indicate 'INITIATE' Request

Prototype:

```
#define V1SL_C2_INITIATE(_CON_ID,_INITIATE_PTR)
```

By calling this output macro, the slave requests a connection establishment from the C2 firmware user.
The user has to acknowledge the service via the input function *v1sl_c2_initiate_done()*.
The slave utilizes the output macro only if the communication channel is open (*v1sl_c2_open_channel()*).

Output Macro:	**V1SL_C2_INITIATE**	
Meaning:	Indicate connection establishment via the C2 firmware	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_CON_ID	Unsigned8 / 000..connection_number-1 (refer to Section 11.3.1 'C2')	Reference of connection whose establishment was requested
_INITIATE_PTR	V1SL_LL_INITIATE_PTR	Pointer to Initiate-REQ-PDU: • _INITIATE_PTR→req (refer to Section 11.3.3 'C2 INITIATE ')
Recommended Actions:		
• Decide on connection acceptance or connection refusal • Prepare response data • Call *v1sl_c2_initiate_done()*		
Corresponding Input Functions:		
v1sl_c2_initiate_done()		

10.2.5.4 Indicate 'ABORT' Request

Prototype:

```
#define V1SL_C2_ABORT(_CON_ID,_ABORT_PTR)
```

By calling this output macro, the slave informs the C2 firmware user of the shut down of the connection identified by the *_CON_ID*. The service is not acknowledged.

Output Macro:		**V1SL_C2_ABORT**
Meaning:	Indicate the 'ABORT' request	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_CON_ID	Unsigned8 / 000..connection_number-1 (refer to Section 11.3.1 'C2')	Connection reference
_ABORT_PTR	V1SL_LL_ABORT_PTR	Pointer to the Abort-PDU: • _ABORT_PTR→abort (refer to Section 11.3.2 'C2 ABORT')
Recommended Actions:		
• Process connection shut down • Establish readiness for accepting a new connection establishment request (via output macro *V1SL_C2_INITIATE()*)		
Corresponding Input Functions:		

10.2.5.5 Acknowledge 'USER ABORT' Request

Prototype:

```
#define V1SL_C2_USER_ABORT_DONE(_CON_ID,_RETURN_VALUE)
```

With this output macro, the slave acknowledges a connection shut down previously initiated by the user (input function *v1sl_c2_user_abort()*).

Output Macro:		**V1SL_C2_USER_ABORT_DONE**
Meaning:	Acknowledge 'USER ABORT' request	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_CON_ID	Unsigned8 / 000..connection_number-1 (refer to Section 11.3.1 'C2')	Reference of connection that was cancelled
_RETURN_VALUE	Unsigned8/ V1SL_OK V1SL_ERR_CON_ID V1SL_ERR_SEQUENCE	Result indication: • Execution OK • Wrong connection reference • Call not allowed in this mode
Recommended Actions:		
• Evaluate transfer parameter • Establish readiness for accepting a new connection establishment request (via *V1SL_C2_INITIATE()*)		
Corresponding Input Functions:		
v1sl_c2_user_abort()		

10.2.5.6 Indicate 'DATA_TRANSPORT' Request

Prototype:

```
#define V1SL_C2_DATA_TRANSPORT(_CON_ID,_DATA_TRANSPORT_PTR)
```

By calling this macro, the slave transfers a Data_Transport-REQ-PDU to the user. The user has to acknowledge the completion of the service by calling the input function *v1sl_c2_data_transport_done()*.
The slave uses the output macro only after the connection has been established (*V1SL_C2_INITIATE()*).

Output Macro:	**V1SL_C2_DATA_TRANSPORT**	
Meaning:	Indicate 'DATA_TRANSPORT' Request	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_CON_ID	Unsigned8 / 000..connection_number-1 (refer to Section 11.3.1'C2')	Connection reference
_DATA_TRANSPORT_PTR	V1SL_LL_DATA_ TRANSPORT_PTR	Pointer to the Data_Transport-REQ-PDU: • _DATA_TRANSPORT_PTR→req (refer to Section 11.3.4 'C2 DATA_TRANSPORT')
Recommended Actions:		
• Process receive data • Make response data available by calling *v1sl_c2_data_transport_done()*		
Corresponding Input Functions:		
v1sl_c2_data_transport_done()		

10.2.5.7 Indicate 'Read Data Set' via the C2-Firmware

Prototype:

```
#define V1SL_C2_READ_DS(_CON_ID,_READ_DS_PTR)
```

By calling this macro, the slave transfers the request for reading a data set to the user. The user has to acknowledge the completion of reading the data set by calling the input function *v1sl_c2_read_ds_done()*.

The slave uses the output macro only after the connection has been established (*V1SL_C2_INITIATE()*).

Output Macro:	**V1SL_C2_READ_DS**	
Meaning:	Indicate request 'Read Data Set' via the C2 Firmware	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_CON_ID	Unsigned8 / 000..connection_number-1 (refer to Section 10.3.1 'C2')	Connection reference to be used for reading the data set
_READ_DS_PTR	V1SL_LL_DS_READ_PTR	Pointer to the Read-REQ-PDU: • _READ_DS_PTR→req (refer to Section 11.4.1 'C1/C2 DS_READ')
Recommended Actions:		
• Evaluate the transfer parameters • Read the specified data set, and enter it in the specified PDU		

• Acknowledge the request by calling *v1sl_c2_read_ds_done()*
Corresponding Input Functions:
v1sl_c2_read_ds_done()

10.2.5.8 Indicate 'Write Data Set' Request via the C2 Firmware

Prototype:

```
#define V1SL_C2_WRITE_DS(_CON_ID,_WRITE_DS_PTR)
```

By calling this macro, the slave transfers a request for writing a data set to the user. The user has to acknowledge the completion of writing the data set by calling the function *v1sl_c2_read_ds_done()*.
The slave uses the output macro only after the connection has been established (*V1SL_C2_INITIATE()*).

Output Macro:		**V1SL_C2_WRITE_DS**
Meaning:	Indicate request 'Write Data Set' via the C2 firmware	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_CON_ID	Unsigned8 / 000..connection_number-1 (refer to Section 11.3.1 'C2')	Connection reference to be used for writing the data set
_WRITE_DS_PTR	V1SL_LL_DS_WRITE_PTR	Pointer to the Write-REQ-PDU: • _WRITE_DS_PTR→req (refer to Section 11.4.2 'C1/C2 DS_WRITE')
Recommended Actions:		
• Evaluate transfer parameter • Write the transferred data set, or route it to the destination component • Acknowledge the request by calling *v1sl_c2_write_ds_done()*		
Corresponding Input Functions:		
v1sl_c2_write_ds_done()		

10.2.6 Interface Expansion of the Output Macros for Multi-Instance Operation

10.2.6.1 General

When their multi-instance capability is activated, all output macros of the V1SL receive one (or two) additional parameter used for referencing the communication channel. Two types of output macros are to be differentiated, in order to clearly describe the meaning of the parameters *..._HANDLE*.
The representation below of the macro-prototypes is only to illustrate the differences in the call parameters in comparison to the non-multi-instance variant of the respective macro. For this reason, the prototype representation is interrupted with '...', and not completely specified.

10.2.6.2 Acknowledge Establishment of a Communication Channel

Prototype:

```
#define V1SL_C0_OPEN_CHANNEL_DONE(...,_V1SL_HANDLE,_HANDLE)
```

```
#define V1SL_C2_OPEN_CHANNEL_DONE(...,_V1SL_HANDLE,_HANDLE)
```

Output Macro:		**V1SL_C0_OPEN_CHANNEL_DONE** **V1SL_C2_OPEN_CHANNEL_DONE**
Meaning:	Acknowledge establishment of a C0 or C2 communication channel	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
...		Parameters that depend on the concrete function
_V1SL_HANDLE	Unsigned8/ 000..254	Handle of the V1SL; the user has to use this handle for subsequent requests to V1SL via this communication channel
_HANDLE	Unsigned8	User handle
Corresponding Input Functions:		
v1sl_c0_open_channel(), v1sl_c2_open_channel()		

10.2.6.3 Acknowledge the Shut Down of a Communication Channel; Result Indication

Prototype:

```
#define V1SL_C0_...(...,_HANDLE)

#define V1SL_AL_...(...,_HANDLE)

#define V1SL_C1_...(...,_HANDLE)

#define V1SL_C2_...(...,_HANDLE)
```

Output Macro:		**V1SL_C0_.../V1SL_AL_.../V1SL_C1_.../V1SL_C2_...**
Meaning:	Acknowledge the shut down of a communication channel, event indication	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
...		Parameters that depend on a concrete function
_HANDLE	Unsigned8	User handle
Corresponding Input Functions:		
V1SL_C0...(), V1SL_AL...(), V1SL_C1...(), V1SL_H...() , V1SL_SC...(), V1SL_S7...(), V1SL_C2...()		

11 Memory Attributes and Data Types

11.1 General Structures and Data Types

11.1.1 General

Memory attributes are intended for optimally fitting the DPV1 slave package into the corresponding environment regarding an individual memory model. An optimized setting is decisive for the generated program and data memory size as well as for the program runtime. If the attributes are not defined, the compiler that is used automatically sets the attributes to the selected basic memory model (for example, small, medium, large).

11.1.2 Base Program Memory Attributes

Attribute Name	Description
V1SL_CODE_ATTR_NEAR	Near program area
V1SL_CODE_ATTR_FAR	Far program area
V1SL_CODE_ATTR_HUGE	Huge program area

11.1.3 Base Data Memory Attributes

Attribute Name	Description
V1SL_DATA_ATTR_NEAR	Near data area
V1SL_DATA_ATTR_FAR	Far data area
V1SL_DATA_ATTR_HUGE	Huge data area

11.1.4 Base Data Types

Data Type Name	Description
UnsignedOpt	Data type with register word width depending on processor
Unsigned8	Unsigned Byte (8 bits)
Unsigned16	Unsigned Short (16 bits)
Unsigned32	Unsigned Long (32 bits)
Signed8	Signed Byte (8 bits)
Signed16	Signed Short (16 bits)
Signed32	Signed Long (32 bits)
Boolean	Is set up as UnsignedOpt; only the values *TRUE* or *FALSE* are allowed

11.1.5 Base Pointer Types

Data Type Name	Description
V1SL_IFA_VOID_PTR	Pointer to a void data with *V1SL_IFA_DATA_ATTR*
V1SL_IFA_UNSIGNED8_PTR	Pointer to an Unsigned8 data with *V1SL_IFA_DATA_ATTR*
V1SL_IFA_UNSIGNED16_PTR	Pointer to an Unsigned16 data with *V1SL_IFA_DATA_ATTR*
V1SL_IFA_UNSIGNED32_PTR	Pointer to an Unsigned32 data with *V1SL_IFA_DATA_ATTR*
V1SL_SYS_VOID_PTR	Pointer to a void data with *V1SL_SYS_DATA_ATTR*
V1SL_SYS_UNSIGNED8_PTR	Pointer to an Unsigned8 data with *V1SL_SYS_DATA_ATTR*
V1SL_SYS_UNSIGNED16_PTR	Pointer to an Unsigned16 data with *V1SL_SYS_DATA_ATTR*
V1SL_SYS_UNSIGNED32_PTR	Pointer to an Unsigned32 data with *V1SL_SYS_DATA_ATTR*
V1SL_LL_VOID_PTR	Pointer to a void data with *V1SL_LL_DATA_ATTR*
V1SL_LL_UNSIGNED8_PTR	Pointer to an Unsigned8 data with *V1SL_LL_DATA_ATTR*
V1SL_LL_UNSIGNED16_PTR	Pointer to an Unsigned16 data with *V1SL_LL_DATA_ATTR*
V1SL_LL_UNSIGNED32_PTR	Pointer to an Unsigned32 data with *V1SL_LL_DATA_ATTR*
V1SL_INT_VOID_PTR	Pointer to a void data with *V1SL_INT_DATA_ATTR*
V1SL_INT_UNSIGNED8_PTR	Pointer to an Unsigned8 data with *V1SL_INT_DATA_ATTR*
V1SL_INT_UNSIGNED16_PTR	Pointer to an Unsigned16 data with *V1SL_INT_DATA_ATTR*
V1SL_INT_UNSIGNED32_PTR	Pointer to an Unsigned32 data with *V1SL_INT_DATA_ATTR*

11.1.6 V1SL Program Memory Attributes

Attribute Name	Description
V1SL_IFA_CODE_ATTR	Program memory attribute of the V1SL input functions to the higher level (user), as well as for callback functions to the user
V1SL_SYS_CODE_ATTR	Program memory attribute of the V1SL input functions to the system environment
V1SL_INT_CODE_ATTR	Internal program memory attribute V1SL (C0, C2 firmware)

11.1.7 V1SL Data Memory Attributes

Attribute Name	Description
V1SL_IFA_DATA_ATTR	Data memory attribute of higher level (user)
V1SL_LL_DATA_ATTR	Data memory attribute of the memory that can be addressed by the PBC
V1SL_SYS_DATA_ATTR	Data memory attribute of the system environment
V1SL_INT_DATA_ATTR	Internal data attribute V1SL (C0, C2 firmware)

11.1.8 V1SL Firmware Version Structure and Pointer

V1SL_STRUC_VERSION (v1sl_get_version())		
Parameter	**Type/Value**	**Description**
components_installed	Unsigned16 / (refer to Section 12.1.2 'ID of Installed V1SL Firmware Components' on page 116) V1SL_COMP_INSTALLED_C0 V1SL_COMP_INSTALLED_SUB_AL V1SL_COMP_INSTALLED_SUB_C1 V1SL_COMP_INSTALLED_C2 V1SL_COMP_INSTALLED_DPC31	Bit field related to the generated firmware components: • State machine of the cyclical (DP standard) services (C0, MSCY1S) • Alarm state machine (AL, MSAL1S) • State machine of acyclical services with the parameterization master (C1, MSAC1S) • State machine of acyclical services with C2 masters (C2, MSAC2S) • PBC DPC31 driver
main_interface	Unsigned8	Version counter for interface changes of the firmware package
function	Unsigned8	Version counter for changes in the functionality of the firmware package
bugfix	Unsigned8	Version counter for error removals
Parameter Type	**Value**	**Description**
V1SL_SYS_VERSION_PTR	V1SL_STRUC_VERSION V1SL_SYS_DATA_ATTR *	Type of pointer to the version data structure

11.1.9 Error Data Structure

V1SL_STRUC_ERROR (V1SL_FATAL_ERROR())		
Parameter	**Type/Value**	**Description**
line	Unsigned16	Line number
module	Unsigned8	Compilation unit
detail_ident	Unsigned8	Error type for selecting a specific error detail information structure within the subsequent union
detail	V1SL_UNION_ERROR_DETAIL	Union with structured error detail information (not further described)

Parameter Type	Value	Description
V1SL_FAR_ERROR_PTR	V1SL_STRUC_ERROR V1SL_DATA_ATTR_FAR *	Type of pointer to the error data structure

11.2 C0 Firmware Structures

11.2.1 C0 Detail Info Structure and Pointer

V1SL_STRUC_C0_DETAIL (V1SL_C0C2_GET_PATH_INFO())		
Parameter	**Type/Value**	**Description**
reserved	Unsigned8	Unused, reserved for future expansions
Parameter Type	**Value**	**Description**
V1SL_SYS_C0_ DETAIL_PTR	V1SL_STRUC_C0_DETAIL V1SL_SYS_DATA_ATTR *	Type of pointer to the C0 detail data structure

11.2.2 C0 Parameter Structure and Pointer

V1SL_STRUC_C0_PARAMETER_SET (v1sl_c0_add())		
Parameter	**Type/Value**	**Description**
c0_sub_components	Unsigned8 / V1SL_C0_SUB_COMPONENTS _AL V1SL_C0_SUB_COMPONENTS _C1	OR operation on available services User wants to utilize alarm services (possible *only* when related firmware part has been generated) User wants to utilize acyclic read and write of data sets (possible *only* when related firmware has been generated)
c0_sub_funct	Unsigned8 / V1SL_SUB_FUNCT_C0_NO_AD D_CHANGE V1SL_SUB_FUNCT_C0_SYNC V1SL_SUB_FUNCT_C0_FREEZ E V1SL_SUB_FUNCT_AL_ALAR M_SAP	OR operation on optional features Address change via the service 'Set-Slave-Address' is *not* supported SYNC mode supported FREEZE mode supported SAP 50 used for alarm handling (possible *only* when related firmware part has been generated)
c0_pno_ident_number_high	Unsigned8	High byte of the PNO identification number of the slave module
c0_pno_ident_number_low	Unsigned8	Low byte of the PNO identification number of the slave module
c0_no_address_change	Unsigned8 / TRUE FALSE	Support for changing the PROFIBUS station address of the slave at runtime with 'Set Slave Address' telegram: • no • yes
c0_sync_supported	Unsigned8 / TRUE FALSE	Support of the slave's SYNC functionality (temporary freezing of the output data) • yes • no
c0_freeze_supported	Unsigned8 / TRUE FALSE	Support of the slave's FREEZE functionality (temporary freezing of the input data) • yes • no
al_user_sequence_mode	Unsigned8 / (refer to Section 12.2.12 'Alarm' on	Alarm mode supported by the user:

	page 122) V1SL_SEQC_MODE_TOTAL_00 V1SL_SEQC_MODE_OFF V1SL_SEQC_MODE_TOTAL_02 V1SL_SEQC_MODE_TOTAL_04 V1SL_SEQC_MODE_TOTAL_08 V1SL_SEQC_MODE_TOTAL_12 V1SL_SEQC_MODE_TOTAL_16 V1SL_SEQC_MODE_TOTAL_24 V1SL_SEQC_MODE_TOTAL_32	• Alarm state machine deactivated; the parameter is to be preassigned with this value if the AL firmware is not generated • Alarm state machine supports type mode only; (one alarm of each type is to be processed at a point in time with the parameterization master) • Alarm state machine supports the sequence mode; 2 alarms maximum of any type may be processed at one point in time with the parameterization master • Sequence mode; 4 alarms... • Sequence mode; 8 alarms... • Sequence mode; 12 alarms... • Sequence mode; 16 alarms... • Sequence mode; 24 alarms... • Sequence mode; 32 alarms...
Note:	The parameters described below determine the slave's memory resources. In each case, they should be **less** or **equal** regarding the parameters equivalent by name when calling the service *pbc_open_device()*.	
c1_alarm_sap_support	Unsigned8 / TRUE FALSE	Support of alarm acknowledgements via the optional SAP 50: • yes (this is possible only if the AL firmware is generated) • no
c0_input_buffer_ptr	V1SL_LL_UNSIGNED8_PTR / NIL otherwise	Pointer to user's input data buffer • Utilization of the exchange buffer system within the V1SL; the user does not have to allocate a buffer • Utilization of the single buffer system; user has to assign buffer of the length *c0_input_buffer_len* , and enter the pointer to it in this parameter (is possible only if the C0-RQB-firmware is generated)
c0_output_buffer_ptr	V1SL_LL_UNSIGNED8_PTR / NIL otherwise	Pointer to user's output buffer • Utilization of the exchange buffer system within the V1SL; the user does not have to allocate a buffer • Utilization of the single buffer system; user has to assign buffer of the length *c0_output_buffer_len* , and enter the pointer to it in this parameter (is possible only if the C0-RQB-firmware is generated)
c0_input_buffer_len	Unsigned8 / 000..244 245..255	Maximum length of the input data buffer: • permissible range • impermissible range
c0_output_buffer_len	Unsigned8 / 000..244 245..255	Maximum length of the output data buffer: • permissible range • impermissible range
c0_user_diag_buffer_len	Unsigned8 / 000..238	Maximum length of the data buffer needed by the **User** (without standard diagnostic length of 6 bytes): • permissible range

	239..255	• impermissible range
c0_ssa_buffer_len	Unsigned8 / 000..003 004..244 245..255	Maximum length of the buffer for 'Set Slave Address' telegrams (the value is relevant only if the parameter *c0_no_address_change = FALSE* is set): • impermissible range • permissible range • impermissible range
c0_prm_buffer_len	Unsigned8 / 000..007 008..244 245..255	Maximum length of the parameterization data buffer: • impermissible range • permissible range • impermissible range
c0_cfg_buffer_len	Unsigned8 / 000 001..244 245..255	Maximum length of configuration data buffer : • impermissible value • permissible range • impermissible range
c1_pdu_buffer_len	Unsigned8 / 000..003 004..244 245..255	Maximum buffer length for data sets via the C1 firmware; the value is to be assigned also if the AL firmware is generated (alarm acknowledgements). The value is to be assigned only if the C1 or the AL firmware is generated: • impermissible range • permissible range • impermissible range
sc_filter_table_len	Unsigned8 / 000	Maximum length of the table, which defines the filters used for the subscriber functionality (what data from which publisher are relevant for operation) • subscriber not supported
Parameter Type	**Value**	**Description**
V1SL_IFA_C0_ PARAMETER_PTR	V1SL_STRUC_C0_ PARAMETER_SET V1SL_IFA_DATA_ATTR *	Type of pointer to the C0 parameter structure

11.2.3 C0 Slave Address Data Structure and Pointer

		V1SL_STRUC_SSA (V1SL_C0_NEW_SSA())
Parameter	**Type/Value**	**Description**
slave_address	Unsigned8 / 000..125 126..255	New PROFIBUS station address of the slave • Possible range • Impossible range
pno_ident_high	Unsigned8	High byte of the PNO identification number of the slave module
pno_ident_low	Unsigned8	Low byte of the PNO identification number of the slave module
no_address_change	Unsigned8 000 001..255	Value that indicates whether an additional slave address change should be possible • Yes • No
user_data	Unsigned8	User specific data in the GSD file
Parameter Type	**Value**	**Description**
V1SL_LL_SSA_PTR	V1SL_STRUC_SSA V1SL_LL_DATA_ATTR *	Type of pointer to the slave address data structure

11.2.4 C0 Parameterization Data Structure

V1SL_UNION_PRM (V1SL_C0_NEW_PRM())		
Parameter	**Type/Value**	**Description**
byte	V1SL_STRUC_PRM_BYTE	Below in this table
bit	V1SL_STRUC_PRM_BIT	Below in this table
V1SL_STRUC_PRM_BYTE		
Type	**Parameter/Value**	**Description**
Status	Unsigned8	Refer to table 10
wd_fact_1	Unsigned8	
wd_fact_2	Unsigned8	
min_tsdr	Unsigned8	
pno_ident_high	Unsigned8	
pno_ident_low	Unsigned8	
group_ident	Unsigned8	
dpv1_status_1	Unsigned8	Refer to table
dpv1_status_2	Unsigned8	
dpv1_status_3	Unsigned8	
user_data	Unsigned8	
V1SL_STRUC_PRM_BIT		
Parameter	**Type/Value**	**Description**
wd_on	bit	Refer to table 10
freeze_req	bit	
sync_req	bit	
unlock_req	bit	
lock_req	bit	
wd_base_1ms	bit	
fail_safe	bit	
dpv1_enable	bit	
check_cfg_mode	bit	
enable_update_alarm	bit	
enable_status_alarm	bit	
enable_manufacturer_alarm	bit	
enable_diagnostic_alarm	bit	
enable_process_alarm	bit	
enable_pull_plug_alarm	bit	
alarm_mode	bit	
Parameter Type	**Value**	**Description**
V1SL_LL_PRM_PTR	V1SL_UNION_PRM V1SL_LL_DATA_ATTR *	Type of pointer to the slave parameterization data structure

11.2.5 C0 Input/Output Data Lengths Structure and Pointer

V1SL_UNION_IN_OUT_CALC (v1sl_c0_calc_in_out_len())		
Parameter	**Type/Value**	**Description**
cfg_data	struct consists of:	Substructure to be assigned by the user prior to function entry
ptr	V1SL_LL_UNSIGNED8_PTR	Pointer to configuration data , whichare used as the base for the calculation of the length for the input and output area
len	Unsigned8	Length of the configuration data [byte]
data_len	struct consists of:	Substructure assigned by V1SL at function exit (input information is dno longer available!)
input	Unsigned8	Evaluated length of the input data area

output	Unsigned8	Evaluated length of the output data area
Parameter Type	**Value**	**Description**
V1SL_IFA_IN_OUT_ CALC_PTR	V1SL_UNION_IN_OUT_CALC V1SL_IFA_DATA_ATTR *	Type of pointer to the input/output data length structure

11.2.6 C0 Output Data Info Structure and Pointer

V1SL_STRUC_OUTPUT_INFO (v1sl_c0_get_output_info())		
Parameter	**Type/Value**	**Description**
ptr	V1SL_LL_UNSIGNED8_PTR	Pointer to a buffer with output data
state	Unsigned8 / The user can get each information through AND operation with the following values (refer to Section 12.2.7')	Status of the data in the output data buffer as well as additional information (refer also to Section 6.6
	V1SL_OUTPUT_STATE_ DATA	• If this bit is set, the current configuration contains an output data length unequal to 0.
	V1SL_OUTPUT_STATE_ NEW	• If this bit is set, the output data buffer transferred with the element *ptr* contains new output data; the length is as specified in the current configuration sent by the master.
	V1SL_OUTPUT_STATE_ CLEAR	• If this bit is set, the user has to interpret the output buffer as cleared; the output data to which the element *ptr* points are not really equal 0 in every case. If this bit is set, the user should not process the output data.
	V1SL_OUTPUT_STATE_ GC_CLEAR V1SL_OUTPUT_STATE_ GC_UNCLEAR	• These bit values inform the user about the status of the 'Global Control' command that was received last.
Parameter Type	**Value**	**Description**
V1SL_IFA_OUTPUT_ INFO_PTR	V1SL_STRUC_OUTPUT_INFO V1SL_IFA_DATA_ATTR *	Type of the pointer to the output data info structure

11.2.7 C0 Diagnostic Data Union and Pointer

V1SL_UNION_DIAG_PTR (v1sl_c0_set_diag())		
Type	**Parameter/Value**	**Description**
byte_ptr	V1SL_IFA_UNSIGNED8_PTR	Pointer to byte field (unstructured)
struc_ptr	V1SL_IFA_DIAG_PTR	Pointer to union with diagnostic data (structured, below in this table)
V1SL_UNION_DIAG		
Type	**Parameter/Value**	**Description**
rev	V1SL_STRUC_REV_DIAG	below in this table
ken	V1SL_STRUC_KEN_DIAG	below in this table
chn	V1SL_STRUC_CHN_DIAG	below in this table
dev	V1SL_STRUC_DEV_DIAG	below in this table
status	V1SL_STRUC_STATUS_DIAG	below in this table
alarm	V1SL_STRUC_ALARM_DIAG	below in this table
V1SL_STRUC_REV_DIAG		
Type	**Parameter/Value**	**Description**
sign_revision	Unsigned8	refer to Section 6.4.3

V1SL_STRUC_KEN_DIAG		
Type	**Parameter/Value**	**Description**
sign_len	Unsigned8	refer to Section 6.4.4
slots[]	Unsigned8	
V1SL_STRUC_CHN_DIAG		
Type	**Parameter/Value**	**Description**
sign_ident	Unsigned8	refer to Section 6.4.5
number	Unsigned8	
code	Unsigned8	
V1SL_STRUC_DEV_DIAG		
Type	**Parameter/Value**	**Description**
sign_len	Unsigned8	refer to Section6.4.6
user_data	Unsigned8	
V1SL_STRUC_STATUS_DIAG		
Type	**Parameter/Value**	**Description**
sign_len	Unsigned8	refer to Section 6.4.7
status_type	Unsigned8	
slot_number	Unsigned8	
specifier	Unsigned8	
user_data	Unsigned8	
V1SL_STRUC_ALARM_DIAG		
Type	**Parameter/Value**	**Description**
sign_len	Unsigned8	refer to Section 6.4.8
alarm_type	Unsigned8	
slot_number	Unsigned8	
specifier	Unsigned8	
user_data	Unsigned8	
Parameter Type	**Value**	**Description**
V1SL_IFA_DIAG_PTR	V1SL_UNION_DIAG V1SL_IFA_DATA_ATTR *	Type of pointer to the diagnostic data union

11.2.8 AL Alarm Data Structure

V1SL_STRUC_ALARM (v1sl_al_set_alarm()/V1SL_AL_ALARM_ACK())		
Parameter	**Type/Value**	**Description**
alarm_type	Unsigned8 / (refer to Section 12.2.12 'Alarm on page 122) V1SL_ALARM_TYPE_DIAG V1SL_ALARM_TYPE_PROC V1SL_ALARM_TYPE_PULL V1SL_ALARM_TYPE_PLUG V1SL_ALARM_TYPE_STAT V1SL_ALARM_TYPE_UPDT V1SL_ALARM_TYPE_ MANU_MIN ... V1SL_ALARM_TYPE_ MANU_MAX	Specify the alarm event: • Diagnostic alarm • Process alarm • Pull alarm • Plug alarm • Status alarm • Update alarm • Permissible value range for manufacturer-specific alarm types
slot_number	Unsigned8 / 000 ...	Slot which triggered the alarm; the value should correspond to the number assigned with the configuration data sent by the master • Permissible range

	V1SL_SLOT_NUMBER_MAX	
specifier	Unsigned8 / (refer to Section 12.2.12 'Alarm on page 122) V1SL_SPEC_SEQC_MASK V1SL_SPEC_SEQC_START_BIT V1SL_SPEC_ADD_ACK_MASK V1SL_SPEC_ALARM_SPEC_ MASK V1SL_SPEC_ALARM_SPEC_ SLOT_ERR_VALUE V1SL_SPEC_ALARM_SPEC_ NO_ERR_VALUE	Additional alarm information: • The user can enter the sequence number of the alarm in these bits. The sequence number can be used in each alarm mode (type or sequence mode). • First bit of the sequence number in the element *specifier*; it is intended for encoding the sequence number via shift operations (*seqc_num* << *V1SL_SPEC_SEQC_START_BIT*). • Through OR operation of this value with the element *specifier*, the user can request additional acknowledgement information about the alarm from the parameterization master; for example, with the service 'Write Data set'. • The user can add alarm information in these bits; for this, an OR operation of one and/or two of the following values with the element *specifier* is required: • 'Slot has error' • 'Error removed'; if 'Slot has error' is also activated, additional errors are active.
user_data_len	Unsigned8 / (refer to Section 12.2.12 'Alarm on page 122) 000 ... V1SL_AL_ USER_DATA_LEN_MAX	Length of the user-specific data of the alarm: • Permissible range; the value 0 is not commonly used, but it is not rated as an error.
user_data_ptr	V1SL_IFA_UNSIGNED8_PTR	Pointer to a user-specific data block of the alarm; the user has to allocate the block with at least the length *user_data_len*
Parameter Type	**Value**	**Description**
V1SL_IFA_ALARM_PTR	V1SL_STRUC_ALARM V1SL_IFA_DATA_ATTR *	Type of pointer to the alarm data structure

11.3 C2 Firmware Structures

11.3.1 C2 Detail Info Structure and Pointer

V1SL_STRUC_C2_DETAIL (V1SL_C0C2_GET_PATH_INFO())		
Type	**Parameter/Value**	**Description**
poll_timeout	Unsigned16 / (refer to Section 12.4.1Poll Timeout Value' V1SL_POLL_TIMEOUT_ MIN ... V1SL_POLL_TIMEOUT_	Characteristic of the slave. During this time, the slave is able to respond to a master request at least with an Idle PDU. The value to be specified is a factor which will be multiplied with the time base of 10ms • Permissible range

	MAX	
pdu_size	Unsigned8 / (refer to Section 12.4.4 'Parameter user_data_len for Data Set Operations') 000 ... V1SL_C1C2_ USER_DATA_LEN_MAX	Largest permissible net data length that is to be transferred view within a PDU from the Layer4 • permissible range
connection_number	Unsigned8 / (refer to Section 25.1) 001 ... V1SL_CFG_C2_ CONNECTION_NUMBER_ MAX	Number of C2 connections that can be utilized parallel via the communication channel • permissible range
final_communication_ point	Boolean / TRUE FALSE	This parameter is relevant only when using the standard interface. It identifies whether the user application is • a communication endpoint or • a link
auto_generate_ initiate_resp	Boolean / TRUE FALSE	This parameter is relevant only when using the standard interface • The C2 firmware itself generates the response to an Initiate request. Via *V1SL_C2_INITIATE()*, the response data structure is transferred to the user. The user decides whether it wants to accept the connection (unchanged return of the PDU) or reject it (generate an Abort-PDU). • The parameter is to be set only if the parameter *final_communication_point* is set also. • The user is responsible for generating the response to an Initiate request.
profile_features_ supported_1	Unsigned8	These parameters are relevant only if the parameter *auto_generate_initiate_resp* is set. With these parameters, the user identifies the values that are to be set for the Initiate response (refer also to section 11.3.3 'C2 INITIATE on page 109).
profile_features_ supported_2	Unsigned8	
profile_ident_ number_low	Unsigned8	
profile_ident_ number_high	Unsigned8	
Parameter Type	**Value**	**Description**
V1SL_SYS_C2_ DETAIL_PTR	V1SL_STRUC_C2_DETAIL V1SL_SYS_DATA_ATTR *	Type of pointer to the C2 detail data structure

11.3.2 C2 ABORT Data Structure and Pointer

		V1SL_STRUC_ABORT_REQ
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 / (refer to Section 12.4.1 'Parameter function_number for Data Set Operations')	Function ID; if the user triggers the 'Abort' service , the user has to assign this element.
subnet	Unsigned8 / (refer to Section 12.3.3 'Parameter subnet at an Abort -PDU')	Identification of the causing hardware component; if the user triggers the 'Abort' service , the user has to assign this element.
reason_code	Unsigned8 / (refer to Section 0 '	Identification of cause for cancellation, and responsible FW component; if the user triggers the 'Abort' service, the user has to

	Parameter reason_code of an Abort-PDU')	assign this element.
Type	**Parameter/Value**	**Description**
V1SL_LL_ABORT_PTR	V1SL_STRUC_ABORT_REQ V1SL_LL_DATA_ATTR *	Type of pointer to the 'Abort' data structure

11.3.3 C2 INITIATE Data Structure and Pointer

V1SL_UNION_INITIATE (V1SL_C2_INITIATE())		
Parameter	**Type/Value**	**Description**
req	V1SL_STRUC_INITIATE_REQ	below in this table
res	V1SL_STRUC_INITIATE_RES	below in this table
nrs	V1SL_STRUC_INITIATE_NRS	below in this table
abort	V1SL_STRUC_ABORT_REQ	refer to Section 11.3.2 'C2 ABORT)
V1SL_STRUC_INITIATE_REQ		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 / (refer to Section 12.4.1 'Parameter function_number for Data Set Operations')	Function ID; does not have to be filled in by the user
reserved_1	Unsigned8 / 0x00	No evaluation by user
reserved_2	Unsigned8 / 0x00	No evaluation by user
reserved_3	Unsigned8 / 0x00	No evaluation by user
send_timeout_high send_timeout_low	Unsigned8 / Unsigned8 / V1SL_POLL_TIMEOUT_MIN ... V1SL_POLL_TIMEOUT_MAX	Monitoring time recommendation by the master (time base 10ms); during this time, the master has to fetch the response data provided by the slave; does not have to be filled in by the user: • Permissible value range of (*send_timeout_high*<<8)+ (*send_timeout_low*)
features_supported_1	Unsigned8 /	Identifies the features of the master's C2 component; does not have to be filled in by the user.
features_supported_2	Unsigned8 / (refer to Section 12.3.5 'Parameter features_supported of an Initiate-PDU')	
profile_features_ supported_1 profile_features_ supported_2	Unsigned8 / Unsigned8 / (refer to Section 12.3.6 ' Parameter profile_features_supported of an Initiate-PDU')	Identifies the features of the master's C2 user component; does not have to be filled in by the user.
profile_ident_number_high	Unsigned8	Manufacturer's profile ID, input of the C2 master; does not have to be filled in by the user.
profile_ident_number_low	Unsigned8	
add_address_parameter	Unsigned8 / (refer to Section 12.3.7 ' add_address_parameter of an Initiate-PDU')	Layer 7 address; does not have to be filled in by the user.

V1SL_STRUC_INITIATE_RES		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8	Function ID; unchanged since request
max_len_data_len	Unsigned8 / 000 ... pdu_size (for recommended value, refer to 11.3.1 'C2)	Maximum PDU length in bytes on the Layer7 level that may be used via this connection. User makes this entry: • permissiblele value range • The value is not to exceed *pdu_size* that is set at *v1sl_c2_open_channel()* • Regarding the low limit, exact specifications not finished yet.
features_supported_1 features_supported_2	Unsigned8 / Unsigned8 / (refer to Section 12.3.5 'Parameter features_supported of an Initiate-PDU')	Identifies the features of the slave's C2 component; is entered by the C2 firmware after receipt of the response data.
profile_features_ supported_1 profile_features_ supported_2	Unsigned8 / Unsigned8 / (refer to Section 0 ' Parameter profile_features_supported of an Initiate-PDU')	Identifies the features of the slave's C2 user component; user has to make this entry.
profile_ident_number_high	Unsigned8	Manufacturer's profile ID of the slave with C2 capability; entry is to be made by the user; S7 protocol is identified with 0x8000
profile_ident_number_low	Unsigned8	
add_address_parameter	Unsigned8 / (refer to Section 0 ' add_address_parameter of an Initiate-PDU' on page 125)	Layer 7 address; entry is to be made by the user.

V1SL_STRUC_INITIATE_NRS		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 / (refer to Section 12.4.1 'Parameter function_number for Data Set Operations')	Function ID; entry is to be made by the user.
error_decode	Unsigned8 / (refer to Section 12.4.5 'Parameter error_decode for Data Set Operations')	Identifies specific meaning of the elements *error_code_1* and *error_code_2*; entry is to be made by the user.
error_code_1	Unsigned8 / (refer to Section 12.4.6 'Parameter error_code1 for Data Set Operations')	Error Code 1; entry is to be made by the user.
error_code_2	Unsigned8 / (refer to Section 12.4.5 'Parameter error_decode for Data Set Operations')	Error Code 2; entry is to be made by the user.
Parameter Type	**Value**	**Description**
V1SL_LL_INITIATE_PTR	V1SL_UNION_INITIATE V1SL_LL_DATA_ATTR *	Type of pointer to the 'Initiate' data structure

11.3.4 C2 DATA_TRANSPORT Data Structure and Pointer

V1SL_UNION_DATA_TRANSPORT (V1SL_C2_DATA_TRANSPORT())		
Parameter	**Type/Value**	**Description**
req	V1SL_STRUC_ DATA_TRANSPORT_REQ	below in this table
res	V1SL_STRUC_ DATA_TRANSPORT_RES	below in this table
nrs	V1SL_STRUC_ DATA_TRANSPORT_NRS	below in this table
abort	V1SL_STRUC_ABORT_REQ	refer to Section 11.3.2 'C2 ABORT' on page 108

V1SL_STRUC_DATA_TRANSPORT_REQ		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 / (refer to Section 12.4.1 'Parameter function_number for Data Set Operations')	Function ID; does not have to be filled in by the user.
slot_number	Unsigned8 / (refer to Section 12.4.2 'Parameter slot_number for Data Set Operations')	Number of slot (module); does not have to be filled in by the user.
index	Unsigned8 / (refer to Section 12.4.3 'Parameter index for Data Set Operations')	Number of the data set; does not have to be filled in by the user.
user_data_len	Unsigned8 / (refer to Section 12.4.4 'Parameter user_data_len for Data Set Operations')	Length of net data sent to the slave by the master; does not have to be filled in by the user.
user_data	Unsigned8	First byte of net data sent to the slave by the mater; does not have to be filled in by the user.

V1SL_STRUC_DATA_TRANSPORT_RES		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8	Function ID; unchanges since request
slot_number	Unsigned8 / (refer to Section 12.4.2 'Parameter slot_number for Data Set Operations')	Number of slot (module); entry is to be made by the user.
index	Unsigned8 / (refer to Section 12.4.3 'Parameter index for Data Set Operations')	Number of the data set; entry is to be made by the user.
user_data_len	Unsigned8 / (refer to Section 12.4.4 'Parameter user_data_len for Data Set Operations')	Length of user data to be transferred to the master; entry is to be made by the user.
user_data	Unsigned8	Data regarding *index* and *slot_number*; number of byte is equal to *user_data_len*.

V1SL_STRUC_DATA_TRANSPORT_NRS		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 / (refer to Section 12.4.1 'Parameter function_number for Data Set Operations')	Function ID; entry is made by the user.
error_decode	Unsigned8 / (refer to Section 12.4.5 'Parameter error_decode for Data Set	Identifies the specific meaning of the elements *error_code_1* and *error_code_2*; entry is to be made by the user.

	Operations')	
error_code_1	Unsigned8 / (refer to Section 12.4.6 'Parameter error_code1 for Data Set Operations')	Error Code 1; entry is to be made by the user.
error_code_2	Unsigned8 / (refer to Section 12.4.5 'Parameter error_decode for Data Set Operations')	Error Code 2; entry is to be made by the user.
Parameter Type	**Value**	**Description**
V1SL_LL_DATA_TRANSPORT_PTR	V1SL_UNION_DATA_TRANSPORT V1SL_LL_DATA_ATTR *	Type of pointer to the 'Data Transport' structure

11.4 C1/C2 Structures (Joint Utilization)

11.4.1 C1/C2 DS_READ Data Structure and Pointer

V1SL_UNION_DS_READ (V1SL_C1_READ_DS()/V1SL_C2_READ_DS())		
Parameter	**Type/Value**	**Description**
req	V1SL_STRUC_DS_READ_REQ	below in this table
res	V1SL_STRUC_DS_READ_RES	below in this table
nrs	V1SL_STRUC_DS_READ_NRS	below in this table
abort	V1SL_STRUC_ABORT_REQ	refer to Section 11.3.2 'C2 ABORT on page 108
V1SL_STRUC_DS_READ_REQ		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 / (refer to Section 12.4.1 'Parameter function_number for Data Set Operations')	Function ID; does not have to be filled in by the user.
slot_number	Unsigned8 / (refer to Section 12.4.2 'Parameter slot_number for Data Set Operations')	Number of slot (module); does not have to be filled in by the user.
index	Unsigned8 / (refer to Section 12.4.3 'Parameter index for Data Set Operations')	Number of the data set; does not have to be filled in by the user.
user_data_len	Unsigned8 / (refer to Section 12.4.4 'Parameter user_data_len for Data Set Operations')	Length of net data that is requested; does not have to be filled in by the user.
V1SL_STRUC_DS_READ_RES		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8	Function ID; unchanged since request
slot_number	Unsigned8	Number of slot (module); unchanged since request
index	Unsigned8	Number of data set; unchanged since request
user_data_len	Unsigned8	Length data that was read by the user; to be entered by the user
	Unsigned8	Starting with this element, the user enters the requested data; the length [byte] is equal to *user_data_len*.
V1SL_STRUC_DS_READ_NRS		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 /	Function ID; is to be entered by the user.

	(refer to Section 12.4.1 'Parameter function_number for Data Set Operations)	
error_decode	Unsigned8 / (refer to Section 12.4.5 'Parameter error_decode for Data Set Operations)	Identifies the specific meaning of the elements *error_code_1* and *error_code_2*; is to be entered by the user.
error_code_1	Unsigned8 / (refer to Section 12.4.6Parameter error_code1 for Data Set Operations')	Error Code 1; is to be entered by the user.
error_code_2	Unsigned8 / (refer to Section 12.4.5 'Parameter error_decode for Data Set Operations')	Error Code 2; is to be entered by the user.
Parameter Type	**Value**	**Description**
V1SL_LL_DS_READ_ PTR	V1SL_UNION_DS_READ V1SL_LL_DATA_ATTR *	Type of pointer to the 'Read Data Set' structure

11.4.2 C1/C2 DS_WRITE Data Structure and Pointer

V1SL_UNION_DS_WRITE (V1SL_C1_WRITE_DS()/V1SL_C2_WRITE_DS())		
Parameter	**Type/Value**	**Description**
req	V1SL_STRUC_DS_WRITE_REQ	below in this table
res	V1SL_STRUC_DS_WRITE_RES	below in this table
nrs	V1SL_STRUC_DS_WRITE_NRS	below in this table
abort	V1SL_STRUC_ABORT_REQ	refer to Section 11.3.2 'C2 ABORT on page 108
V1SL_STRUC_DS_WRITE_REQ		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 / (refer to Section 12.4.1 'Parameter function_number for Data Set Operations')	Function ID; does not have to be filled in by the user.
slot_number	Unsigned8 / (refer to Section 12.4.2 'Parameter slot_number for Data Set Operations')	Number of slot (module); does not have to be filled in by the user.
index	Unsigned8 / (refer to Section 12.4.3 'Parameter index for Data Set Operations')	Number of the data set; does not have to be filled in by the user.
user_data_len	Unsigned8 / (refer to Section 12.4.4 'Parameter user_data_len for Data Set Operations')	Length of net data that was sent to the slave; does not have to be filled in by the user.
user_data	Unsigned8	First byte of the net data sent to the slave; does not have to be filled in by the user.
V1SL_STRUC_DS_WRITE_RES		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8	Function ID; unchanged since request
slot_number	Unsigned8	Number of the slot (module); unchanged since request
index	Unsigned8	Number of the data set; unchanged since request
user_data_len	Unsigned8	User enters length of the processed data.

V1SL_STRUC_DS_WRITE_NRS		
Parameter	**Type/Value**	**Description**
function_number	Unsigned8 / (refer to Section 12.4.1 'Parameter function_number for Data Set Operations')	Function ID; is to be entered by the user
error_decode	Unsigned8 / (refer to Section 12.4.5 'Parameter error_decode for Data Set Operations')	Identifies the specific meaning of the elements *error_code_1* and *error_code_2*; is to be entered by the user.
error_code_1	Unsigned8 / (refer to Section 12.4.6 'Parameter error_code1 for Data Set Operations')	Error Code 1; is to be entered by the user.
error_code_2	Unsigned8 / (refer to Section 12.4.5 'Parameter error_decode for Data Set Operations')	Error Code 2; is to be entered by the user.
Parameter	**Type/Value**	**Description**
V1SL_LL_DS_WRITE_ PTR	V1SL_UNION_DS_WRITE V1SL_LL_DATA_ATTR *	Type of pointer to the 'Write Data Set' structure

12 Encoding Rules

12.1 General Values

12.1.1 ID of V1SL Return Values and Error Messages

return_value, _RETURN_VALUE		
Symbolic Value	Numeric Value	Description
V1SL_ERR_FATAL	0x00	After a fatal error is signalled (output macro *V1SL_FATAL_ERROR()*), the V1SL returns this value to the user at each V1SL function call.
V1SL_OK	0x01	Successful execution of a V1SL function (synchronous).
V1SL_OK_ASYNC	0x02	Successful execution of a V1SL function (asynchronous).
V1SL_OK_EOM	0x03	Successful execution of a V1SL function; the receive data was copied completely to the specified buffer.
V1SL_ERR_RESOURCE	0x80	Resources are insufficient for operating the communication channel. The connection endpoint was not set up, or no connection resources are available.
V1SL_ERR_PATH	0x81	The call of *V1SL_C0C2_GET_PATH_INFO()* was acknowledged negative.
V1SL_ERR_LOWER_LAYER	0x82	Establishment the communication channel to the PBC driver was acknowledged negative.
V1SL_ERR_REF	0x83	The specified C2 connection reference is invalid.
V1SL_ERR_SEQUENCE	0x84	Command not allowed for the current slave mode.
V1SL_ERR_LOCAL_ABORT	0x85	The C2 connection reference was cancelled locally.
V1SL_ERR_REMOTE_ABORT	0x86	The remote partner has cancelled the connection.
V1SL_ERR_PROTOCOL	0x87	When the connection was established, a protocol error occurred, or the connection was cancelled because of a protocol error by the remote partner.
V1SL_ERR_TIMEOUT	0x88	When the connection was established, the monitoring time expired, or an existing connection was cancelled because the monitoring time expired.
V1SL_ERR_OPCODE	0x89	Wrong element opcode in the case of a user request at the request block interface to the C0/C2 firmware.
V1SL_ERR_HANDLE	0x8A	No more free handle/communication channel available.
V1SL_ERR_INT_DATA	0x8B	Allocation of internal data memory was acknowledged negative.
V1SL_ERR_DP_STATE	0x8D	Command not allowed for current DP mode.
V1SL_ERR_AL_STATE	0x8E	Command not allowed for the current state of the alarm state machine.
V1SL_ERR_SSA_STATE	0x8F	Command not allowed for the current state of the 'Set Slave Address' state machine.
V1SL_ERR_REAL_CFG_STATE	0x90	Command not allowed for current state of 'Get_Cfg' state machine (expected configuration).

V1SL_ERR_TARGET_CFG_STATE	0x91	Command not allowed for current state of the 'Check_Cfg' state machine (configuration sent by the master).
V1SL_ERR_APP_STATE	0x92	Command not allowed for current state of the 'Application Ready' state machine.
V1SL_ERR_INPUT_STATE	0x93	Command not allowed for current state of the input data state machine.
V1SL_ERR_SAP_STATE	0x94	Command not allowed for current SAP mode.
V1SL_ERR_DIAG_BUFFER	0x95	Diagnostic elements or diagnostic length wrong.
V1SL_ERR_PARAMETER	0x96	The value(s) of one (several) transfer parameter(s) is(are) not in the specified range.
V1SL_ERR_QUEUE	0x97	The alarm queue is disabled; the function was called in the context of the output macro *V1SL_AL_ALARM_ACK()*.
V1SL_ERR_USER_PRM_DATA	0x99	Wrong user parameterization data.
V1SL_ERR_USER_PRM_DATA_LEN	0x9A	Wrong user parameterization data.
V1SL_ERR_CFG_DATA	0x9B	The configuration data is syntactically wrong, or it exceeds the maximum user data length specified by the user in *v1sl_c0_add()*.
V1SL_ERR_CFG_SKF_LEN	0x9C	The length of the configuration data is a multiple of the expected SKF format.
V1SL_ERR_CFG_SLOT_NUMBER	0x9D	The number of slots to be changed exceeds the number specified by the user.
V1SL_ERR_CFG_SKF_SIGN	0x9E	Error in SKF
V1SL_ERR_CFG_AKF_SIGN	0x9F	Error in AKF
V1SL_ERR_REAL_CFG_PTR	0xA0	It was not possible to fetch a buffer for the expected configuration data.
V1SL_ERR_INPUT_PTR	0xA1	It was not possible to fetch a buffer for the input data.
V1SL_ERR_ADDRESS	0xA2	This value is not routed to the user.

12.1.2 Identification of Installed V1SL Firmware Components

components_installed (V1SL_STRUC_VERSION)		
Symbolic Value	Numeric Value	Description
V1SL_COMP_INSTALLED_C0	0x0001	C0 firmware; State machine of the cyclical (DP standard) services (MSCY1S)
V1SL_COMP_INSTALLED_SUB_AL	0x0004	AL firmware; Alarm state machine (AL, MSAL1S)
V1SL_COMP_INSTALLED_SUB_C1	0x0008	C1 firmware; State machine of the acyclical services with the parameterization master (C1, MSAC1S)
V1SL_COMP_INSTALLED_C2	0x0100	C2 firmware; State machine of the acyclical services with C2 masters (C2, MSAC2S)
V1SL_COMP_INSTALLED_DPC31	0x4000	PBC DPC31 driver firmware

12.1.3 Handle Values

handle/_HANDLE (pbc_open_device()/V1SL_..._OPEN_CHANNEL_DONE())		
Symbolic Value	Numeric Value	Description
V1SL_HANDLE_EMPTY	0xFF	Error when processing the function: • *PBC_open_device()* • At the user interface, the value is

		relevant only in multi-instance operation to the macros *V1SL_C0_OPEN_CHANNEL_DONE()* *V1SL_C2_OPEN_CHANNEL_DONE()*

12.2 C0 Firmware Values

12.2.1 Slave Components

		c0_sub_components (V1SL_STRUC_C0_PARAMETER_SET)
Symbolic Value	Numeric Value	Description
V1SL_C0_SUB_COMPONENTS_AL	0x01	User wants to utilize alarm services (possible *only* when related firmware part has been generated)
V1SL_C0_SUB_COMPONENTS_C1	0x02	User wants to utilize acyclic read and write of data sets (possible *only* when related firmware has been generated)

12.2.2 Optional Slave Features

		c0_sub_funct (V1SL_SYS_PBC_DETAIL_PTR) (V1SL_STRUC_C0_PARAMETER_SET)
Symbolic Value	Numeric Value	Description
V1SL_SUB_FUNCT_C0_SYNC	0x01	SYNC mode supported
V1SL_SUB_FUNCT_C0_FREEZE	0x02	FREEZE mode supported
V1SL_SUB_FUNCT_C0_NO_PUBLISHER	0x04	Publisher not supported
V1SL_SUB_FUNCT_C0_NO_ADD_CHANGE	0x08	Address change not supported
V1SL_SUB_FUNCT_AL_ALARM_SAP	0x20	SAP 50 is used for alarm handling

12.2.3 Slave Control Parameters

		mode (v1sl_c0_control())
Symbolic Value	Numeric Value	Description
V1SL_CONTROL_START	0x01	With this command, the slave goes from the DP mode *V1SL_DP_STATE_OFF* to the DP mode *V1SL_DP_STATE_NO_DATA_EX* and can then be parameterized by a master. This control sequence must be carried out after establishment of a communication channel, and setting up the C0 slave's memory resources. The call is possible only in the DP mode *V1SL_DP_STATE_OFF*.

V1SL_CONTROL_STOP	0x02	With this command, the slave enters the DP mode *V1SL_DP_STATE_OFF* and after that, can't be parameterized by any master. It can be restarted via the slave control with *V1SL_CONTROL_START*. The call is possible only in the DP mode *V1SL_DP_STATE_NO_DATA_EX* or *V1SL_DP_STATE_DATA_EX* . Parameterization data (*V1SL_C0_NEW_PRM()*) and configuration data sent by the master (*V1SL_C0_NEW_CFG()*) that are processed at the time of the call are no longer valid (no more buffer accesses !) It is also not allowed for the user to acknowledge these services to the slave!
V1SL_CONTROL_LEAVE_MASTER	0x03	With this command, the user can reset an active slave to the DP mode *V1SL_DP_STATE_NO_DATA_EX* (for example, when the expected configuration changes). The slave will request new parameterization by the master. The call is possible only in the DP mode *V1SL_DP_STATE_NO_DATA_EX* or *V1SL_DP_STATE_DATA_EX* .
V1SL_CONTROL_SSA_DONE	0x04	With this command, the user informs the slave that processing a 'Set Slave Address' telegram is completed (*V1SL_C0_NEW_SSA()*).
V1SL_CONTROL_PRM_OK	0x05	With this command, the user informs the slave of having successfully completed processing received parameterization data (*V1SL_C0_NEW_PRM()*).
V1SL_CONTROL_PRM_ERROR	0x07	With this command, the user rejects a wrong parameterization data telegram (*V1SL_C0_NEW_PRM()*).
V1SL_CONTROL_CFG_OK	0x08	With this command, the user informs the slave of the successful comparison of received configuration data sent by the master with its expected configuration (*V1SL_C0_NEW_CFG()*).
V1SL_CONTROL_CFG_UPDATE	0x09	With this command, the user informs the slave of the successful comparison of received configuration data sent by the master with its expected configuration (*V1SL_C0_NEW_CFG()*). Also the new configuration data sent by the master will be available for other stations as the expected configuration of the slave from now on.
V1SL_CONTROL_CFG_ERROR	0x0A	With this command, the user rejects a wrong configuration data sent by the master message (*V1SL_C0_NEW_CFG()*).

V1SL_CONTROL_APP_READY	0x0B	With this command, the user informs the slave that it is ready to receive user data. The call is always required after the receipt of a configuration sent by the master telegram in the DP mode *V1SL_DP_STATE_NO_DATA_EX*. As long as the call is not made but the slave has already reached the DP mode *V1SL_DP_STATE_DATA_EX* , the slave is in the static diagnostic mode on the bus-side (refer to Section 6.5).

12.2.4 DP Watchdog States

		_STATE (V1SL_C0_WD_STATE_REPORT())
Symbolic Value	Numeric Value	Description
V1SL_WD_STATE_BAUD_SEARCH	0x40	Watchdog is in the baudrate search mode
V1SL_WD_STATE_BAUD_CONTROL	0x80	Watchdog is in the baudrate monitoring mode
V1SL_WD_STATE_DP_MODE	0xC0	Watchdog is in the DP mode; that means the DP data traffic with the parameterization master is monitored.

12.2.5 DP Modes

		_STATE (V1SL_C0_DP_STATE_REPORT())
Symbolic Value	Numeric Value	Description
V1SL_DP_STATE_INVALID	0x00	Initial mode; is signalled to the user only when the C0 firmware request block interface is used.
V1SL_DP_STATE_OFF	0x01	Slave is not active on the bus-side and responds to the master requests with 'Service not activated' (RS) on all SAPs.
V1SL_DP_STATE_NO_DATA_EX	0x02	Slave is not in the mode of cyclical data exchange; generally, this is the parameterization and configuration phase when the communication between the parameterization master and the slave is established.
V1SL_DP_STATE_DATA_EX	0x03	Slave is in the cyclical data exchange mode with the parameterization master.

12.2.6 Bus Error LED States

		_STATE (V1SL_C0_LED_STATE_REPORT())
Symbolic Value	Numeric Value	Description
V1SL_LED_STATE_OFF	0x04	BF LED is to be switched off.
V1SL_LED_STATE_ON	0x08	BF LED is to be switched on.
V1SL_LED_STATE_FLASH	0x0C	BF LED is to blink at a frequency specified by the system.

12.2.7 Parameters for Parameterization

		DPV1 Status Masks (V1SL_UNION_PRM)
Symbolic Value	Numeric Value	Description
V1SL_DPV1_STATUS_1_	0x80	Mask for extracting the slave's

OP_MODE_MASK		'Operation Mode' from the DPV1 Status Byte 1
V1SL_DPV1_STATUS_2_ CFG_FAULT_MASK	0x01	Mask for extracting the slave's configuration mode from the DPV1 Status Byte 2; this value is also supplied to the user with new configuration data (*V1SL_C0_NEW_CFG()*).
V1SL_DPV1_STATUS_2_ ALARM_TYPE_MASK	V1SL_ALARM_TYPE_ DIAG_VALUE | V1SL_ALARM_TYPE_ PROC_VALUE | V1SL_ALARM_TYPE_ PUPL_VALUE | V1SL_ALARM_TYPE_ UPDT_VALUE | V1SL_ALARM_TYPE_ STAT_VALUE | V1SL_ALARM_TYPE_ MANU_VALUE	Mask for extracting the slave's permissible alarm types from the DPV1 Status Byte 2; this value is also supplied to the user when the alarm state machine is started (*V1SL_AL_STATE_REPORT()*).
V1SL_DPV1_STATUS_3_ ALARM_MODE_MASK	0x07	Mask for extracting the slave's alarm mode from the DPV1 Status Byte 3; this value (in recalculated form) is also supplied to the user when the alarm state machine is started (*V1SL_AL_STATE_REPORT()*).
V1SL_DPV1_STATUS_3_ PRM_CMD_MASK	0x80	Mask to determine whether a PRM command is contained in the user parameterization data; also the slave processes this bit internally if the H-function module is activated.

12.2.8 Configuration Parameters

_MODE (V1SL_C0_NEW_CFG())		
Symbolic Value	Numeric Value	Description
V1SL_CFG_MODE_STOP_ON_FAULT	0x01	Configuration sent by master and expected configuration have to agree 100% so that a startup can be performed.
V1SL_CFG_MODE_RUN_ON_FAULT	0x02	Deviations at comparison between configuration sent by master/expected configuration are tolerated; a startup can be performed.

12.2.9 States of the Output Data Buffer

state (V1SL_STRUC_OUTPUT_INFO)		
Symbolic Value	Numeric Value	Description
V1SL_OUTPUT_STATE_DATA	0x02	If this bit is set, the configured output data length is NOT equal to 0.
V1SL_OUTPUT_STATE_NEW	0x04	If this bit is set, the output data buffer transferred with the element *ptr* contains new output data. The length corresponds to the length specified in the current configuration sent by the master.
V1SL_OUTPUT_STATE_CLEAR	0x08	If this bit is set, the user is to interpret the output data buffer as deleted; the output data

		to which the element *ptr* points are not really 0 in every case. If this bit is set, the user should not process the output data.
V1SL_OUTPUT_STATE_GC_ CLEAR	0x10	These bit informs the user about the state of the 'Global Control' command received last.
V1SL_OUTPUT_STATE_GC_ UNCLEAR	0x20	

12.2.10 Diagnostic Control

		diag_control (v1sl_c0_set_diag())
Symbolic Value	Numeric Value	Description
V1SL_EXT_DIAG_SET	0x01	Set bit 'extended diagnostic'.
V1SL_EXT_DIAG_RESET	0x00	Reset bit 'extended diagnostic'.
V1SL_EXT_DIAG_UNCHANGE	0x02	Don't influence bit 'extended diagnostic'.
V1SL_EXT_DIAG_OVF_SET	0x04	Set bit 'extended diagnostic data overflow'.
V1SL_EXT_DIAG_OVF_RESET	0x00	Reset bit 'extended diagnostic data overflow'.
V1SL_EXT_DIAG_OVF_UNCHANGE	0x08	Don't influence bit 'extended diagnostic data overflow'.
V1SL_STAT_DIAG_SET	0x10	Set bit 'static diagnostic'; this is possible only in a compatibility mode of the slave (*v1sl_c0_control()*).
V1SL_STAT_DIAG_RESET	0x00	Reset bit 'static diagnostic'.
V1SL_STAT_DIAG_UNCHANGE	0x20	Don't influence bit 'static diagnostic'.
V1SL_SEND_DIAG_WITH_ALARM	0x40	Diagnostics will only be sent together with any alarm; if necessary, the slave waits for the next alarm set by the user; the C0 firmware resets the bit when leaving the DP mode *V1SL_DP_STATE_DATA_EX* .

12.2.11 Diagnostic Control (Status)

		status_type (V1SL_STRUC_STATUS_DIAG)
Symbolic Value	Numeric Value	Description
V1SL_STATUS_TYPE_SIGN	0x80	This bit always has to be set in the element *status_type*. In addition, the user has to encode one of the following values in the element *status_type* (through OR operation):
V1SL_STATUS_TYPE_ STATUS_MESSAGE	0x01	Status message
V1SL_STATUS_TYPE_ MODUL_STATUS	0x02	Module status
V1SL_STATUS_TYPE_ PRM_COMMAND_ACK	0x1E	PRM Command Acknowledge (only for H-systems)
V1SL_STATUS_TYPE_ H_STATUS_MESSAGE	0x1F	H-status (only for H-systems)
V1SL_STATUS_TYPE_MANU_MIN	0x20	Value range of the manufacturer-specific status types
V1SL_STATUS_TYPE_MANU_MAX	0x7E	

12.2.12 Alarm Control

al_user_sequence_mode, _SEQUENCE_DEPTH (v1sl_c0_add()/V1SL_AL_STATE_REPORT())		
Symbolic Value	Numeric Value	Description
V1SL_SEQC_MODE_TOTAL_00	0x00	Alarm state machine is deactivated; no alarm is to be transmitted or set by the user.
V1SL_SEQC_MODE_OFF	0x01	Alarm state machine operates in the type mode only; that means, one alarm each of each type may be processed at a point in time with the parameterization master
V1SL_SEQC_MODE_TOTAL_02	0x02	Sequence mode; 2 alarms of any type may be processed with the parameterization master at a point in time
V1SL_SEQC_MODE_TOTAL_04	0x04	Sequence mode; 4 alarms...
V1SL_SEQC_MODE_TOTAL_08	0x08	Sequence mode; 8 alarms...
V1SL_SEQC_MODE_TOTAL_12	0x0C	Sequence mode; 12 alarms...
V1SL_SEQC_MODE_TOTAL_16	0x10	Sequence mode; 16 alarms...
V1SL_SEQC_MODE_TOTAL_24	0x18	Sequence mode; 24 alarms...
V1SL_SEQC_MODE_TOTAL_32	0x20	Sequence mode; 32 alarms...

alarm_type (V1SL_STRUC_ALARM)		
Symbolic Value	Numeric Value	Description
V1SL_ALARM_TYPE_DIAG	0x01	Diagnostic alarm
V1SL_ALARM_TYPE_PROC	0x02	Process alarm
V1SL_ALARM_TYPE_PULL	0x03	Pull alarm
V1SL_ALARM_TYPE_PLUG	0x04	Plug alarm
V1SL_ALARM_TYPE_STAT	0x05	Status alarm
V1SL_ALARM_TYPE_UPDT	0x06	Update alarm
V1SL_ALARM_TYPE_MANU_MIN	0x20	Value range of manufacturer-specific alarm types
V1SL_ALARM_TYPE_MANU_MAX	0x7E	

specifier (V1SL_STRUC_ALARM)		
Symbolic Value	Numeric Value	Description
V1SL_SPEC_SEQC_MASK	0xF8	In the bits of this mask, the user can enter the sequence number of the alarm. The sequence number can be used in any alarm mode (type or sequence mode).
V1SL_SPEC_SEQC_START_BIT	0x03	First bit of the sequence number in the element *specifier* which is intended for encoding the sequence number via shift operations (*seqc_num* << *V1SL_SPEC_SEQC_START_BIT*).
V1SL_SPEC_ADD_ACK_MASK	0x04	Through OR operation of this value with the element *specifier*, the user can request additional acknowledgement information for the alarm from the parameterization master; e.g. by using the service 'Write Data Set'.
V1SL_SPEC_ALARM_SPEC_MASK	0x03	The user can add additional alarm information in the bits of this mask; for this, an OR operation of one and/or two of the following values with the element *specifier* is required:
V1SL_SPEC_ALARM_SPEC_ SLOT_ERR_VALUE	0x01	• 'Slot has error'
V1SL_SPEC_ALARM_SPEC_ NO_ERR_VALUE	0x02	• 'Error removed'; If, in addition, 'Slot has error' is still active, other errors are active.

user_data_len (V1SL_STRUC_ALARM)		
Symbolic Value	Numeric Value	Description
V1SL_AL_USER_DATA_LEN_MAX	0x3B	Maximum length of the net data of the alarm

_ALARM_TYPE_BIT_FIELD, alarm_type_bit_field (V1SL_AL_STATE_REPORT()/v1sl_al_withdraw_alarm())		
Symbolic Value	Numeric Value	Description
V1SL_ALARM_TYPE_DIAG_VALUE	0x20	Diagnostic alarms are permissible/are to be deleted.
V1SL_ALARM_TYPE_PROC_VALUE	0x40	Process alarms are permissible/are to be deleted.
V1SL_ALARM_TYPE_PUPL_VALUE	0x80	Pull/plug alarms are permissible/are to be deleted.
V1SL_ALARM_TYPE_STAT_VALUE	0x08	Status alarms are permissible/are to be deleted
V1SL_ALARM_TYPE_UPDT_VALUE	0x04	Update alarms are permissible/are to be deleted.
V1SL_ALARM_TYPE_MANU_VALUE	0x10	Manufacturer-specific alarms are permissible/are to be deleted.
V1SL_ALARM_TYPE_NONE_VALUE	0x00	No alarms are permissible/are to be deleted.
V1SL_ALARM_TYPE_ALL_VALUE	0xFF	All alarms are permissible/are to be deleted.

sequence_number (v1sl_al_withdraw_alarm())		
Symbolic Value	Numeric Value	Description
V1SL_SEQUENCE_NUMBER_MIN	0x00	Value range of the alarm sequence numbers
V1SL_SEQUENCE_NUMBER_MAX	0x1F	
V1SL_SEQUENCE_NUMBER_ALL	0xFF	Alarms of all sequence numbers are to be deleted.

12.3 C2 Firmware Values

12.3.1 Poll Timeout Values

poll_timeout (V1SL_STRUC_C2_DETAIL)		
Symbolic Value	Numeric Value	Description
V1SL_POLL_TIMEOUT_MIN	0x0001	Minimum possible poll time
V1SL_POLL_TIMEOUT_MAX	0x7FFF	Maximum possible poll time

12.3.2 Channel Type

channel_type (V1SL_STRUC_C2_DETAIL)		
Symbolic Value	Numeric Value	Description
V1SL_CHANNEL_LOCAL	0x01	Communication channel that ends locally in the C2 firmware.
V1SL_CHANNEL_DEVICE	0x02	Device-oriented communication channel that extends to the PBC driver.

12.3.3 Parameter subnet at an Abort -PDU

subnet (V1SL_STRUC_ABORT_REQ)		
Symbolic Value	Numeric Value	Description
V1SL_SUBNET_LOCAL	0x01	Module is directly connected to PROFIBUS DP
V1SL_SUBNET_REMOTE	0x02	Module is connected to the bus via an

		IM/Link

12.3.4 Parameter reason_code of an Abort-PDU

In the parameter *reason_code* of an Abort-PDU, the following information is encoded:

- the cause for the shut down of the connection
- firmware component that triggered the shut down

The reasons for the shut down for which the C2 firmware user is responsible are valid for one profile ID respectively (corresponding to the 'Initiate' parameter *profile_ident_number*). The reasons for the shut down of the user layer, listed in the table below, are valid for:

- *profile_ident_number*: Value 0x8000 (S7 protocol).

Encoding	Component that triggers the connection shut down	Meaning of error
0x01	Layer 2	UE
0x03	Layer 2	RS
0x09	Layer 2	NR
0x0A	Layer 2	DH
0x0C	Layer 2	RDL
0x0D	Layer 2	RDH
0x11	MSAC_C2	ABT_SE: sequence error
0x12	MSAC_C2	ABT_FE: received invalid request
0x13	MSAC_C2	ABT_TO: timeout
0x14	MSAC_C2	ABT_RE: received invalid response
0x15	MSAC_C2	ABT_IV: invalid service of the DP/T user
0x17	MSAC_C2	ABT_IA: invalid address information
0x1F	MSAC_C2	ABT_RS: no resources
0x2B	MSAC_C2 User	Layer7 protocol error
0x2C	MSAC_C2 User	Invalid address information in the Initiate-PDU
0x2D	MSAC_C2 User	Regular connection cancellation
0x2E	MSAC_C2 User	Communication restart for technical reasons
0x2F	MSAC_C2 User	No resources

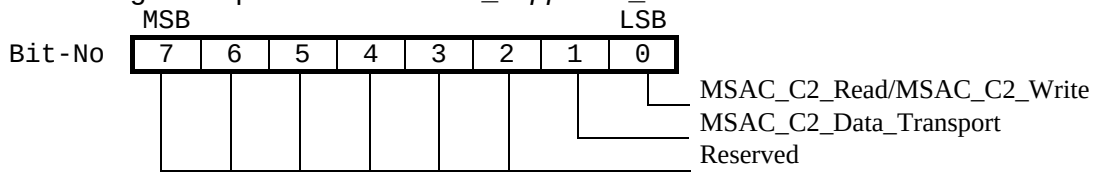
Table 13: Encoding of the Parameter *reason_code* of an Abort-PDU

12.3.5 Parameter features_supported of an Initiate-PDU

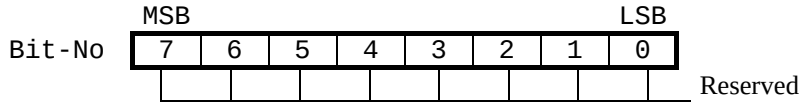
The parameter *features_supported* identifies the performance of the slave's C2 component. It is structured in the form of two octet strings of eight bits each. The two figures below show the meaning of the individual bit positions. The slave in the present implementation fulfils the functionality 'MSAC2_Read' / 'MSAC2_Write' and 'MSAC2_Data_Transport'. Thus, the parameters are assigned as follows:

- *features_supported_1*: Value 0x03,
- *features_supported_2*: Value 0x00.

Encoding of the parameter *features_supported_1*:



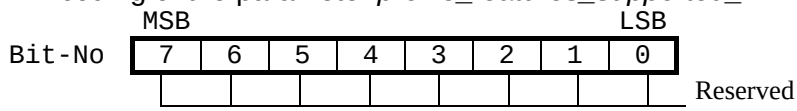
Encoding of the parameter *features_supported_2*:



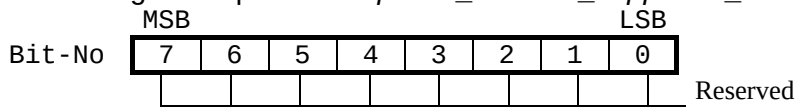
12.3.6 Parameter profile_features_supported of an Initiate-PDU

The parameter *profile_features_supported* identifies the performance of the user layer of the slave's C2 component . It is structured in the form of two octet strings of 8 bits each. The meaning of the individual bit positions is shown in the two figures below.

Encoding of the parameter *profile_features_supported_1*:



Encoding of the parameter *profile_features_supported_2*:



12.3.7 add_address_parameter of an Initiate-PDU

The source and destination address is encoded in this parameter on the level of Layer7.

add_address_parameter (V1SL_UNION_INITIATE)		
Parameter	**Type/Value**	**Description**
s_type	Unsigned8 / 000 001	Identification of the structure of *s_addr*: • TSAP • Complete network address
s_len	Unsigned8	Length of *s_addr*
d_type	Unsigned8 / 000 001	Identification of the structure of *d_addr*: • TSAP • Complete network address
d_len	Unsigned8	Length of *d_addr*
s_addr	See below	Source address: TSAP or sender's complete network address
d_addr	See below	Destination address: TSAP or receiver's complete network address

s_addr, d_addr (V1SL_UNION_INITIATE)				
Parameter	**stype**	**Type**	**Subparameter**	**Description**
s_addr	0	Unsigned8	api	Both parameters in conjunction form the *tsap_id* in the S7 protocol (ProfileIdentNumber = 0x8000).
		Unsigned8	scl	
	1	Unsigned8	api	Both parameters in conjunction form the *tsap_id* in the S7 protocol (ProfileIdentNumber = 0x8000).
		Unsigned8	scl	
		Octet-String[6]	network_address	Subnet_Id of sender
		Octet-String[s_len-8]	mac_address	MAC address of sender
d_addr	0	Unsigned8	api	Both parameters in conjunction form the

		Unsigned8	scl	*tsap_id* in the S7 protocol (ProfileIdentNumber = 0x8000).
	1	Unsigned8	api	Both parameters in conjunction form the *tsap_id* in the S7 protocol (ProfileIdentNumber = 0x8000).
		Unsigned8	scl	
		Octet-String[6]	network_address	Subnet_Id of receiver
		Octet-String[d_len-8]	mac_address	MAC address of receiver

12.4 C1/C2 Values

12.4.1 Parameter function_number for Data Set Operations

Function Basis		
Symbolic Value	Numeric Value	Description
V1SL_FN_BASE	0x40	This value is the basis for generating all function numbers; see below
Function Error Basis		
Symbolic Value	Numeric Value	Description
V1SL_FN_ERROR_BASE	0x80	The user has to add the element *function_number* to this basis, in the case an error was detected during the execution of a service.
Function Encoding		
Symbolic Value	Numeric Value	Description
		These values are to be added to *V1SL_FN_BASE* to obtaim the function number of the following service:
V1SL_FC_INITIATE	0x17	• Connection establishment
V1SL_FC_ABORT	0x18	• Shut down of connection
V1SL_FC_DATA_TRANSPORT	0x11	• Data transport
V1SL_FC_DS_READ	0x1E	• Read data set
V1SL_FC_DS_WRITE	0x1F	• Write data set
function_number (V1SL_UNION_DS_READ/ V1SL_UNION_DS_WRITE V1SL_UNION_DATA_TRANSPORT)		
Symbolic Value	Symbolic Value Combination	Description
		These values identify the type of service in a request telegram that is made available to the user via corresponding output macros:
V1SL_FN_INITIATE	V1SL_FN_BASE + V1SL_FC_INITIATE	• Connection establishment
V1SL_FN_ABORT	V1SL_FN_BASE + V1SL_FC_ABORT	• Connection shut down
V1SL_FN_DATA_TRANSPORT	V1SL_FN_BASE + V1SL_FC_DATA_TRANSPORT	• Data transport
V1SL_FN_DS_READ	V1SL_FN_BASE + V1SL_FC_DS_READ	• Read data set
V1SL_FN_DS_WRITE	V1SL_FN_BASE + V1SL_FC_DS_WRITE	• Write data set

12.4.2 Parameter slot_number for Data Set Operations

		slot_number (V1SL_STRUC_..._REQ/ V1SL_STRUC_..._RES)
Symbolic Value	Numeric Value	Description
V1SL_SLOT_NUMBER_MAX	0xFE	Maximum number of a slot/module

12.4.3 Parameter index for Data Set Operations

		Index (V1SL_STRUC_..._REQ/ V1SL_STRUC_..._RES)
Symbolic Value	Numeric Value	Description
V1SL_INDEX_MAX	0xFE	Maximum number or highest index of a data set

12.4.4 Parameter user_data_len for Data Set Operations

		user_data_len (V1SL_STRUC_..._REQ/ V1SL_STRUC_..._RES)
Symbolic Value	Numeric Value	Description
V1SL_C1C2_ USER_DATA_LEN_MAX	0xF0	Maximum length of net data of a data set operation

12.4.5 Parameter error_decode for Data Set Operations

		error_decode (V1SL_STRUC_..._NRS)
Symbolic Value	Numeric Value	Description
V1SL_ED_DPV1	0x80	Error message that the user of the DPV1 slave triggered. In this case, the additional element *error_code_1* is to be encoded as described in Section 12.4.6 'Parameter error_code1 for Data Set Operations' on page 127 . The element *error_code_2* can be assigned specific to the user.
V1SL_ED_PROFIBUS_FMS	0xFE	Error message that was triggered by PROFIBUS FMS. The additional elements *error_code_1* and *error_code_2* are to be assigned corresponding to the specifications of PROFIBUS FMS.
V1SL_ED_HART	0xFF	Error message that was returned by a HART®[1] module. The additional elements *error_code_1* and *error_code_2* are to be assigned corresponding to the HART® protocol.

12.4.6 Parameter error_code1 for Data Set Operations

Note: The values described in the table below are to be encoded in the element *error_code_1* of an NRS PDU if, in the element *error_decode* of the same PDU, the value *V1SL_ED_DPV1* was specified!

[1] HART® is a registered trademark of the HART Communication Foundation

		error_code_1 (V1SL_STRUC_..._NRS)
Symbolic Value	Numeric Value	Description
V1SL_EC1_DPV1_CLASS_ APPLICATION	0xA0	Error class: 'Application'; this value is to be expanded details (by OR operation) • Read error • Write error • Module error • Version conflict • Functionality not supported • Value range of the manufacturer-specific error IDs
V1SL_EC1_DPV1_CODE_ READ	0x00	
V1SL_EC1_DPV1_CODE_ WRITE	0x01	
V1SL_EC1_DPV1_CODE_ MODULE	0x02	
V1SL_EC1_DPV1_CODE_ VERSION	0x08	
V1SL_EC1_DPV1_CODE_ NOT_SUPPORTED	0x09	
V1SL_EC1_DPV1_CODE_USER_MIN	0x0A	
V1SL_EC1_DPV1_CODE_USER_MAX	0x0F	
V1SL_EC1_DPV1_CLASS_ ACCESS	0xB0	Error class: 'Access'; this value is to be expanded with details (by OR operation) • Wrong indication of the element *index* • Wrong length indication of the data *user_data_len* that is to be written • Wrong indication of the element *slot_number* • Type conflict • Faulty data area • State conflict • Access denied • Impermissible range • Impermissible parameter • Impermissible type • Value range of the manufacturer-specific error IDs
V1SL_EC1_ACC_CODE_ INDEX_INVALID	0x00	
V1SL_EC1_ACC_CODE_ WRITE_LEN	0x01	
V1SL_EC1_ACC_CODE_ SLOT_INVALID	0x02	
V1SL_EC1_ACC_CODE_ TYPE_CONFLICT	0x03	
V1SL_EC1_ACC_CODE_ AREA_INVALID	0x04	
V1SL_EC1_ACC_CODE_ STATE_CONFLICT	0x05	
V1SL_EC1_ACC_CODE_ ACCESS_DENIED	0x06	
V1SL_EC1_ACC_CODE_ RANGE_INVALID	0x07	
V1SL_EC1_ACC_CODE_ PARAMETER_INVALID	0x08	
V1SL_EC1_ACC_CODE_ TYPE_INVALID	0x09	
V1SL_EC1_ACC_CODE_USER_MIN	0x0A	
V1SL_EC1_ACC_CODE_USER_MAX	0x0F	
V1SL_EC1_DPV1_CLASS_ RESOURCE	0xC0	Error class: 'Resource'; this value is to be expanded with details (by OR operation) • Conflict when reading the data • Conflict when writing the data • Resource currently busy • Resource not available • Value range of the manufacturer-specific error IDs
V1SL_EC1_RES_CODE_ READ_CONSTRAIN	0x00	
V1SL_EC1_RES_CODE_ WRITE_CONSTRAIN	0x01	
V1SL_EC1_RES_CODE_ BUSY	0x02	
V1SL_EC1_RES_CODE_ UNAVAILABLE	0x03	
V1SL_EC1_RES_CODE_USER_MIN	0x08	
V1SL_EC1_RES_CODE_USER_MAX	0x0F	
V1SL_EC1_DPV1_CLASS_ USER_MIN	0xD0	Value range of the manufacturer-specific error classes
V1SL_EC1_DPV1_CLASS_ USER_MAX	0xF0	

13 Resources

13.1 General

The tables below provide component-granular information about the needed program and data memory requirement of the C0 and C2 firmware. Among other things, this requirement depends on the following:

- on the tool set used
- on the selection of single or multi-instance operation
- on the assignment of the C0/C2 output macros
- on the assignment of the system output macros
- on the selection of memory attributes
- on additional configuration switches (for example, static or dynamic memory management)

For these reasons, only basic values (minimum/maximum) are indicated for memory requirements.
The user only has to take the memory of the all components into consideration, which will be generated (refer to Section 25.1)
The stated memory requirement identifies the amount that is needed per utilized C0 or C2 instance (communication channel).

Note: To determine the program and data memory requirement as well as the needed timers of the complete slave firmware, the values in Section 21 are to be taken into account.

13.2 System Interface

These memory shares have to be included in the overall memory requirement for each V1SL generation.

	System Interface V1SL
Keil C51	...
Tasking C166	>0,3
Borland C++	...
MS Visual C++	...

Table 14: Program Memory Requirement System Interface V1SL Part (in KBytes)

	System Interfacee V1SL
Keil C51	...
Tasking C166	>2
Borland C++	...
MS Visual C++	...

Table 15: Data Memory Requirement System Interface V1SL Part (in Bytes)

13.3 C0 Firmware

	RQB-IFA	C0	AL	C1	S7
Keil C51	...	>3,8	>2,9	>1,6	>2,0
Tasking C166	>1,4 (near) >2,0 (huge)	>3,8 (near) >5 (huge)	>2,2 (near) >4,0 (huge)	>1,8 (near) >2,2 (huge)	>1,6 (near) >2,0 (huge)
Borland C++	...	...	...	...	...
MS Visual C++	...	...	...	...	...

Table 16: Program Memory Requirement C0 Firmware (in KBytes)

The stated data memory requirement identifies the amount that is needed per utilized slave instance (communication channel).

	RQB-IFA	C0	AL	C1	S7
Keil C51	...	>60	>60	>10	>10
Tasking C166	>22	>60 (near) >70 (huge)	>60	>10	>10
Borland C++	...	...	...	...	...
MS Visual C++	...	...	...	...	...

Table 17: Data Memory Requirement C0 Firmware (in Bytes)

13.4 C2 Firmware

	C2
Keil C51	>...
Tasking C166	>5,4 (near) >7,1 (huge)
Borland C++	...
MS Visual C++	...

Table 18: Program Memory Requirement C2 Firmware (in KBytes)

The stated data memory requirement identifies the amount that is needed per utilized slave instance (communication channel).

	C2
Keil C51	...
Tasking C166	>40+26*number of connections
Borland C++	...
MS Visual C++	...

Table 19: Data Memory Requirement C2 Firmware (in Bytes)

14 General

To implement DP and DPV1 slave solutions, PROFIBUS controllers (PBC) are used on the bus-side. A higher level firmware module (in this case, this refers to the C0/C2 firmware) requires hiding specific features of the PBC used. This requirement is met with the PBC driver firmware described below.

15 System Integration

15.1 System Prerequisites

These systems require only a minimum of prerequisites, since the PBC has integrated the processing of the complete protocols that are necessary to the slave.

15.2 Initialization

The system environment, after initializing the V1SL (*v1sl_init()*), has to perform a hardware reset for each PBC DPC31. In addition, a memory test of the internal DPC31 RAM is recommended. After that, the system environment has to announce each PBC DPC31 to the PBC driver firmware (*pbc_open_device()*).

The V1SL is then ready to open communication channels of the C0/C2 firmware.

15.3 Event Handling

In the case of events, the PBC DPC31 generates a hardware interrupt. The events are processed with PBC interrupt priority within the PBC driver's interrupt handler called by the system environment (*pbc_dpc31_int_handler()*).

Note:	Utilizing the DPC31 HW interrupt is not absolutely required. Depending on the performance parameters of the slave module to be designed, solutions are possible where all PBC events are polled in the low-priority sequence level. In that case, the interrupt handler of the PBC driver (*pbc_dpc31_int_handler()*) is to be called cyclically. The high-priority DPC31 hardware interrupt is not used.

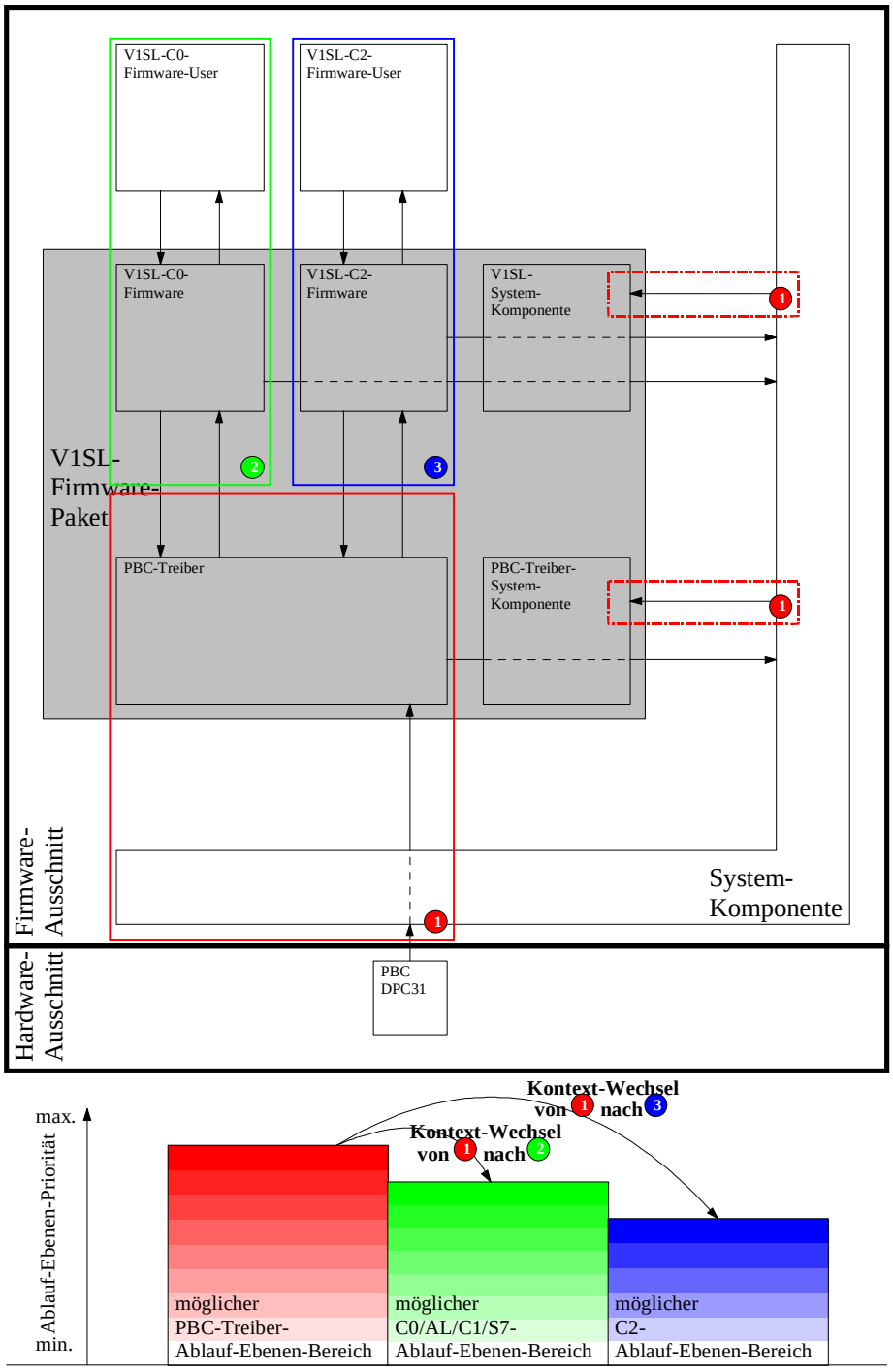
15.4 Sequence Level Configuration and Context Change PBC Driver/C0/C2 Firmware

The description in this section is of interest to systems of which require the following:

- Utilizing the PBC interrupt in the slave module
- Processing the C0 and/or C2 firmware parts of the V1SL in the slave module is required on sequence levels with a priority that differs from the sequence level of the PBC interrupt.

For users to whom these requirements don't apply, notes are provided at the end of this section.

Figure 12 below shows the possible sequence levels within the V1SL firmware package.



<<translation of terms: **Paket** = package; **Treiber** = driver; **Ausschnitt** = segment; **Ablauf-Ebenen-Priorität** = sequence level priority; **Kontext-Wechsel** = context change; **von** = from; **nach** = to; **möglicher PBC-Treiber-Ablaufebenen-Bereich** = possible PBC driver sequence level area>>

Figure 12: Design of the Sequence Layers of V1SL

The area bordered red (1) identifies the firmware parts processed on the priority of the PBC interrupt sequence level.

The areas bordered by a broken red line (1) have to be called at the same priority as the PBC interrupt sequence level. This is to be ensured by the system environment.

The area bordered green (2) represents the sequence level of the C0 firmware and the corresponding user application.

The area bordered blue (3) includes the sequence level of the C2 firmware, and the corresponding user application.

The diagram shown in Figure 1 is to demonstrate as an example how the assignment of the sequence levels to the processing priorities is possible. The color shadings in the three areas identify the priorities from which input function calls of the corresponding firmware part are permitted. As an example, the C0 firmware part is briefly described (2). The events of the C0 firmware part are always indicated to the associated user application with the highest priority of the green area. The call of the C0 input functions, on the other hand, can be located in the priority area of the entire green color shadings.

Therefore, if a context change is necessary, the priority of the PBC interrupt has to be higher than that of the C0 and C2 firmware sequence levels. The priorities of the C0 and C2 firmware parts are independent of each other, and may therefore be the same or different.

The output macros of the PBC driver *PBC_C0C1_EVENT_INDICATION()* and *PBC_C2_EVENT_INDICATION()* are used for connecting the PBC driver and the C0/C2 firmware, and are made available to the user for configuring the context change as needed. In general, the functionality looks like this:

- Within the output macros, the system environment stores the parameters Event Bit Field (*_EVENT_BIT_FIELD*) and Handle (*_HANDLE*) in a system queue. In addition, further (low priority) processing of the queue events has to be triggered. E.g. this can be done by triggering a TRAP, with which an additional (low priority) interrupt is activated.
- The processing function within the (low priority) interrupt or the polling reads one event bit field and one handle respectively from the queue, and utilizes the information to call the corresponding V1SL input function *v1sl_c0c1_perform_services()* or *v1sl_c2_perform_services()*.

This mechanism can be simplified depending on the system; for example, if only one communication channel is opened to the C0 or C2 firmware part of V1SL. No queue is needed in that case. The functionality will then look like this:

- Within the output macros, the handle and the event bitfield are stored in a system variable (! through OR operation with the previous value, don't forget to preassign 0). In addition, further (low priority) processing of the event variables has to be triggered (e.g. with the TRAP mentioned above).

- The (low priority) processing function does the following: it inputs the stored event bitfield and the handle under PBC interrupt disable in temporary variables; then it deletes the originals, cancels the PBC interrupt disable, and calls the corresponding V1SL input function *v1sl_c0c1_perform_services()* or *v1sl_c2_perform_services()*.

For systems that don't include the requirements mentioned above, no context change is necessary, and the 'interconnection' of PBC driver and C0/C2 firmware looks like this:

- The output macro of the PBC driver *PBC_C0C1_EVENT_INDICATION()* is to be coupled directly with the corresponding input function of the C0 Firmware *v1sl_c0c1_perform_services()* in the V1SL configuration file.

- The output macro of the PBC driver *PBC_C2_EVENT_INDICATION()* is to be coupled directly with the corresponding input function of the C2 firmware *v1sl_c2_perform_services()* in the V1SL configuration file.

16 Special Mechanisms

16.1 General

In this section, additional functionalities are explained that are not included in DPV1 description, but that are important to the way a slave works. This includes the following:

- Baudrate search and baudrate monitoring
- User watchdog
- User-dependent setting of the minT_{sdr}.

16.2 Baudrate Search and Baudrate Monitoring

16.2.1 General

This PBC driver functionality is particularly important in DPV1 slave modules whose configuration on the bus-side in a distributed communication network can't be preset (for example, compact/modular slaves). These modules are ready to communicate with other bus stations only after the successful automatic search of the baudrate, and after setting it in the slave.

16.2.2 Activation and Parameters

The functionality of baudrate search and baudrate monitoring is active automatically. Further influencing of the functionality by the system environment is limited to indicating a desired monitoring time for a found baudrate (until the return to the baudrate search mode), in order to override presettings of the PBC driver if needed. For this, the element *baud_control* of the detail block is used when calling *pbc_open_device()* (refer to Section 17.2: Announcing the 'PROFIBUS Controller to the PBC Driver' on page 10). The value to be entered in *baud_control* is not a time value, but a root value defined .The time resulting from this is calculated as follows:

$$\mathbf{T_{baud\_control} = 10ms * baud\_control^2}$$

To use the values preset in the PBC driver, the user has to set *baud_control = 0*.

16.2.3 Monitoring Timer

Baudrate search and baudrate monitoring is based on a timer. The state machine that handles it can enter three different states:

- *PBC_WD_STATE_BAUD_SEARCH*: This state indicates that the state machine for baudrate search is in the search mode.
- *PBC_WD_STATE_BAUD_CONTROL*: This state indicates that a baudrate was found, and that it is monitored continuously in connection with a monitoring time (element *baud_control*) in the detail block, or preset values from Table 20). The monitoring time specifies how long the state machine waits after receiving a telegram until it changes to the *PBC_WD_STATE_BAUD_SEARCH* state.

Baudrate (kBaud)	Monitoring Time/ Preset Value (s)
12000	1
6000	1
3000	2
1500	3
500	4
187.5	5
93.75	6
45.45	8
19.2	10
9.6	20

Table 20: Baudrates and Assigned Monitoring Time

- *PBC_WD_STATE_DP_MODE*: If the slave is parameterized by a master, the parameterization data contains information about a DP monitoring time (*wd_fact1* and *wd_fact2*, and information about its activation (*wd_on*). The DP monitoring time is independent on the monitoring time of the baudrate. If the DP monitoring time is activated, the state machine of the baudrate search changes to the *PBC_WD_STATE_DP_MODE* and thus no longer monitors the baudrate. Only in the case the DP monitoring time has expired the state machine of the baudrate search return to the *PBC_WD_STATE_BAUD_CONTROL* mode.

The system environment can determine the current state of the timer via the call *pbc_get_wd_state()* (refer to Section 17.4) 'Reading Out the Watchdog In general, this is required only for the purpose of a status display (for example, via LED).

16.3 User Watchdog

16.3.1 General

The user watchdog has the task of monitoring ‘life signs’ of the system environment (*pbc_trigger_user_wd()*, refer to Section 17.6 ‘Triggering the User Watchdog’. If the user watchdog expires, it effects the bus-side only in case of the following:

- The cyclical state machine of the V1SL is activated (C0 firmware or MSCY1S)
- The cyclical state machine of the V1SL is in the data exchange mode

If the system shows no 'life sign' under these conditions (for example, because of a system crash), the cyclical DPV1 slave state machine automatically exits the data exchange mode. Thus, the parameterization master operating the slave recognizes the error, and can take defined steps.

Note: Monitoring the life sign of the firmware is useful only if all sequence levels of the firmware are included. Since many systems have several sequence levels, it is recommended to trigger the watchdog on the sequence level with the lowest priority.

Note: If, as mentioned in the preconditions above, the cyclical slave state machine is not activated, triggering the user watchdog (*pbc_trigger_user_wd()*) does not generate an error.

16.3.2 Activation and Parameters

The system environment has to set the monitoring time of the user watchdog for each PBC when calling *pbc_open_device()*. To do this, the element *user_wd_value* in the detail block of the PBC driver is used.

The user watchdog can be switched off by indicating *user_wd_value* = 0.

16.3.3 Monitoring Mechanism

User_wd_value represents a time value with a base of 10ms; therefore, if the system does not trigger the user watchdog, the user watchdog will expire after user_wd_value * 10 milliseconds. The user can specify the value of user_wd_value according to the maximum runtime of his program part in the DP mode V1SL_DP_STATE_DATA_EX:

$user\_wd\_value = 2 * T_{runtime\ of\ application} / 10ms$

2...safety factor

16.4 User-Dependent Setting of $minT_{sdr}$

16.4.1 General

$minT_{sdr}$ is one of the parameters that determines the timing on the bus-side for the response behavior of the slave. It specifies how long the slave waits after receiving an SRD telegram before starting to send the response telegram.

The parameter $minT_{sdr}$ is, according to EN 50170, dependent on the baudrate and the functionality (utilizing C0 and/or C2 services) to be operated on the bus.

16.4.2 Activation and Parameters

Experience has shown that no specified guideline fits to all cases. For this reason, the user can adapt the $minT_{sdr}$ if a PBC is announced (*pbc_open_device()*). For this, the data *mintsdr* in the detail block of the PBC driver is used. The figure below describes the possible settings:

$minT_{sdr}$	**Effect**
000..010	Invalid range, the slave does the following: • If the slave is a pure DP slave (generation of the C0 firmware, but no C2 firmware), immediately after finding a baudrate the PBC driver sets a value of *mintsdr = 11*. • If the slave is utilizing C0 and C2 services (generation of C0 and C2 firmware), immediately after finding a baudrate the PBC driver sets a value that depends on the transmission rate. There may be startup difficulties regarding communication between the bus stations, because the chosen value may not fit to the overall system configuration.
0011..0255	Valid range

17 Input Functions

17.1 Overview

Input Function	Description
pbc_open_device	Announce PBC to the PBC driver firmware
pbc_close_device	PBC setup is no longer valid
pbc_get_wd_state	Reads out watchdog state
pbc_get_baudrate	Reads out baudrate
pbc_trigger_user_wd	Triggers user watchdog
pbc_dpc31_int_handler	PBC DPC31 interrupt handler

17.2 Announce PROFIBUS Controller to the PBC Driver

Prototype:

```
Unsigned8 PBC_SYS_CODE_ATTR pbc_open_device (V1SL_SYS_PBC_DETAIL_PTR
detail_ptr, V1SL_SYS_UNSIGNED8_PTR handle_ptr)
```

By calling this function (system environment), each PBC addressed by the V1SL during further firmware processing is made known to the driver firmware. The system environment has to ensure that the addressed PBC was previously reset (hardware reset).
In addition, the memory resources of the PBC driver are set up. This is necessary, since the DPC31 does not permit the explicit setup of the memory resources during runtime because of its architecture.

Input Function:		**pbc_open_device**
Meaning:		Make PROFIBUS controller (PBC) known to the PBC driver
Transfer:		
Parameter	Value Range	Meaning
detail_ptr	(refer to Section 19.5)	The information of the detail pointer describes the PBC to be utilized. The detail pointer has to be identical with the one that is requested later with *V1SL_PBC_GET_PATH_INFO()* for this PBC when a communication channel is opened.
handle_ptr		Pointer to handle value. The value is entered by V1SL and valid *only* in case of the return value *V1SL_OK*. This vlaue must be used for future calls of PBC driver functions (in case of multi device operation *only*).
	000..254 (value the pointer points to)	• *Single Device Operation:* only one PBC available, therefore the value is not of interest • *Multi Device Operation:* handle of the PBC
Return:		
Value Range		Meaning
V1SL_OK		• Function executed successfully
V1SL_ERR_DEVICE_STATE		• Function was already called for this controller or detail_ptr was not initialized
V1SL_ERR_DEVICE_NOT_SUPPORTED		• PBC type not supported by the PBC driver
V1SL_ERR_DEVICE_HARDWARE_STATE		• PBC in state Offline (e.g. error during reset)
V1SL_ERR_SSA_LEN		• Invalid buffer length for service 'Set-Slave-Address'
V1SL_ERR_PRM_LEN		• Invalid buffer length for parameterization data
V1SL_ERR_CFG_LEN		• Invalid buffer length for configuration data

V1SL_ERR_USER_DIAG_LEN	• Invalid buffer length for user diagnostic data
V1SL_ERR_INPUT_OUTPUT_LEN	• Invalid buffer length for input repectively output data
V1SL_ERR_C1C2_SAP_NUMBER	• Invalid amount of SAP's, which can be handled by the PBC at the same time
V1SL_ERR_RESOURCE	• Memory resources not sufficient to support request
	• •
Corresponding output macros:	

17.3 PROFIBUS Controller Setup no longer valid

Prototype:

```
Unsigned8 PBC_SYS_CODE_ATTR pbc_close_device (
V1SL_SYS_PBC_DETAIL_PTR_PTR detail_ptr_ptr)
```

By calling this function, the system environment can reset a previous announcement and setup of a PBC. Then, the PBC can be announced again with new parameters.

Input Function:		**pbc_close_device**
Meaning:		PROFIBUS controller setup no longer valid
Transfer:		
Parameter	Value Range	Meaning
detail_ptr_ptr		Entry of the detail pointer of the related PBC by V1SL, which was given to V1SL when calling *pbc_open_device()* (pointer is no longer of use for the PBC driver).
Return:		
Value Range		Meaning
V1SL_OK V1SL_ERR_DEVICE_STATE		• Function executed successfully • Calling the function is impermissible in the current PBC driver state (e.g. communication channels still open).
		• •
Corresponding output macros:		

17.4 Read the Watchdog State

Prototype:

```
Unsigned8 PBC_SYS_CODE_ATTR pbc_get_wd_state (void)
```

by calling this function, the system environment determines the current state of the 'Baud Control Timer'. A detailed description of the watchdog mechanism is provided in Section 16.2 'Baudrate Search and Baudrate Monitoring' .

Input Function:		**pbc_get_wd_state**
Meaning:		Read the watchdog state
Transfer:		
Parameter	Value Range	Meaning

Return:	
Value Range	Meaning
PBC_WD_STATE_BAUD_SEARCH PBC_WD_STATE_BAUD_CONTROL PBC_WD_STATE_DP_MODE	The PBCs 'Baud Control Timer' is in the state (refer to Section 0 ' Watchdog States' on page 155): • Baudrate search mode • Baudrate monitoring mode • DP mode
Corresponding output macros:	

17.5 Read the Baudrate

Prototype:

```
Unsigned8 PBC_SYS_CODE_ATTR pbc_get_baudrate (void)
```

By calling this function, the system environment can determine the current baudrate on the bus. Additional details regarding baudrate search are provided in Section 16.2 Search and Baudrate Monitoring'.

Input Function:		pbc_get_baudrate
Meaning:	Read the baudrate	
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range	Comment	
PBC_BAUDRATE_12M	• 12 Mbaud	
PBC_BAUDRATE_6M	• 6 Mbaud	
PBC_BAUDRATE_3M	• 3 Mbaud	
PBC_BAUDRATE_1_5M	• 1.5 Mbaud	
PBC_BAUDRATE_500k	• 500 kBaud	
PBC_BAUDRATE_187_5k	• 187.5 kBaud	
PBC_BAUDRATE_93_75k	• 93.75 kBaud	
PBC_BAUDRATE_45_45k	• 45.45 kBaud	
PBC_BAUDRATE_19_2k	• 19.2 kBaud	
PBC_BAUDRATE_9_6k	• 9.6 kBaud	
Corresponding output macros:		

17.6 Trigger the User Watchdog

Prototype:

```
void PBC_SYS_CODE_ATTR pbc_trigger_user_wd (void)
```

By calling this function, the system environment retriggers the user watchdog of a PBC. Details of the PBC-dependent watchdog mechanisms are provided in Section 17.6 ‘Triggering the User Watchdog'..

Input Function:		pbc_trigger_user_wd
Meaning:		Trigger the User Watchdog
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range		Meaning
Corresponding output macros:		

17.7 PBC DPC31 Interrupt Handler

Prototype:

```
void PBC_SYS_CODE_ATTR pbc_dpc31_int_handler (Unsigned8 device_handle)
```

The PBC DPC31 interrupt handler processes the events of the PBC DPC31. After the PBC driver finished execution, it passes the events to the following via corresponding output macros:

- C0 Firmware via PBC_C0C1_EVENT_INDICATION() (refer to Section 0 'Signalling PBC Events to the C0 Firmware' on page 144)
- C2 Firmware via PBC_C2_EVENT_INDICATION() (refer to Section 0 'Signalling PBC Events to the C2 Firmware‘ on page 145)

If the system environment does not provide a PBC interrupt service routine, this function has to be called cyclically (e.g. polling).

Input Function:		pbc_dpc31_int_handler
Meaning:		PBC DPC31 Interrupt Handler
Transfer:		
Parameter	Value Range	Meaning
Return:		
Value Range		Meaning
Corresponding output macros:		

17.8 Interface Expansion of the Input Functions for Multi Device Operation

When recognizing multi device operation, most input functions of the PBC driver receive an additional parameter *handle* used for referencing the respective PBC.
An exception is the following function:

- pbc_open_device(): By calling this function, the system environment determines the handle of the respective PBC.

The presentation of the function prototypes below is only to clarify the differences in the call parameters in comparison to the non-multi-device variant of the function. For this reason, the prototype presentation is interruped with '...' and thus not specified completely.

Prototype:

```
... PBC_SYS_CODE_ATTR pbc_... (Unsigned8 handle)
```

Input Function:		**pbc_...**
Meaning:	Services and interrupt handler of the PBC driver	
Transfer:		
Parameter	Value Range	Meaning
Handle	000..254 V1SL_HANDLE_EMPTY	PBC handle; the system environment received this value from the PBC driver with the announcement of the PBC (*pbc_open_device()*): • valid value • impermissible
Return:		
Value Range		Meaning
		• depends on the concrete function
Corresponding output macros:		

18 Output Macros

18.1 Overview

Input Function	Description
PBC_C0C1_EVENT_INDICATION	Indication of PBC events to the C0 firmware
PBC_C2_EVENT_INDICATION	Indication of PBC events to the C2 firmware
V1SL_PBC_GET_PATH_INFO	Determine a communication path of the PBC driver
V1SL_PBC_RELEASE_PATH_INFO	Release a communication path of the PBC driver

18.2 Indicate PBC Events to the C0 Firmware

Prototype:

```
#define PBC_C0C1_EVENT_INDICATION(_EVENT_BIT_FIELD)
```

With this output macro, the interrupt handlers of the PBC driver (*pbc_dpc31_int_handler()*) indicate to the C0 firmware that new events are present for processing.

Output Macro:		**PBC_C0C1_EVENT_INDICATION**
Meaning:	Indicate PBC events to the C0 firmware	
Transfer:		
Parameter	Type and Value Range	Meaning
_EVENT_BIT_FIELD	Unsigned16	Bit field with the events that occurred
Recommended Actions:		
• Refer to Section 15.4 'Sequence Level Configuration and Context Change PBC Driver/C0/C2 Firmware' Implement context change in whose result the corresponding input function of the C0 firmware *v1sl_c0c1_perform_services()* is called, or		

• Directly connnect output macro with the corresponding input function of the C0 firmeware *v1sl_c0c1_perform_services()* (refer to Section 25.1)
Corresponding input functions:

18.3 Indicate PBC Events to the C2 Firmware

Prototype:

```
#define PBC_C2_EVENT_INDICATION(_EVENT_BIT_FIELD)
```

With this output macro, the interrupt handler of the PBC driver (*pbc_dpc31_int_handler()*) indicates to the C2 firmware that new events are present for processing.

Output Macro:		**PBC_C2_EVENT_INDICATION**
Meaning:	Indicate PBC events to the C2 firmware	
Transfer:		
Parameter	Type and Value Range	Meaning
_EVENT_BIT_FIELD	Unsigned16	Bit field with the events that occurred
Recommended actions:		
• Refer to 15.4 'Sequence Level Configuration and Context Change PBC Driver/C0/C2 Firmware' Implement context change in whose result the corresponding input function of the C2 firmware *v1sl_c2_perform_services()* is called, or • Directly connect output macro with the corresponding input function of the C2 firmware *v1sl_c2_perform_services()* (refer to Section 25..1		
Corresponding input functions:		

18.4 Request Path Information for the PBC Driver

Prototype:

```
#define V1SL_PBC_GET_PATH_INFO(_RETURN_VALUE,_SYS_PATH,
_SYS_PTR_PTR,_DETAIL_PTR_PTR)
```

Corresponding to the LSA model, the output macro determines two pointers from the specified path description (*_SYS_PATH*):

- The system pointer is used to identify the layer below associated with this path. The V1SL does not need it, but it is stored, and transferred to the system environment as parameter in the case of some output macro calls.
- The detail pointer points to the information which records the specifics of the concrete implementation regarding protocol processing. The system component specifies the values of the individual structural elements. The PBC's detail pointer for the requested communication channel has to be identical with the one that was specified previously when the PBC was announced (*pbc_open_device()*). The structure of the detail information valid for the PBC driver firmware is provided in Section 19.5 'Detail Info Structure of the PBC

Driver'.The parameter *_SYS_PATH* is specified via the function *v1sl_c0_open_channel()* or *v1sl_c2_open_channel()*.

Output Macro:	**V1SL_PBC_GET_PATH_INFO**	
Meaning:	Request path information for the PBC driver	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
_RETURN_VALUE	Unsigned16 / V1SL_SYS_PATH_OK	Status of operation (this data is not filled in by the V1SL, but is to be assigned by the system environment): • System and detail information was entered in the transferred pointers for evaluation by the PBC driver. • The V1SL interprets all other values as error.
_SYS_PATH	V1SL_SYS_PATH_TYPE	Communication path information with which the system environment makes the system and detail information for the PBC driver firmware available. The V1SL fills in this data.
_SYS_PTR_PTR	V1SL_SYS_SYSTEM_PTR PBC_INT_DATA_ATTR *	With this, the V1SL transfers a pointer to a pointer. If the acknowledgement is positive, its content is to be assigned by the system environment. If the system environment does not need this data for calling output macros to the system, NIL can be assigned.
_DETAIL_PTR_PTR	V1SL_SYS_PBC_DETAIL_PTR PBC_INT_DATA_ATTR *	With this, the V1SL transfers a pointer to a pointer. If the acknowledgement is positive, its content is to be assigned detail information of the communication path for the PBC driver firmware by the system environment.
Recommended actions:		
•		
Corresponding input functions:		

18.5 Release Path Information for the PBC Driver

Prototype:

```
#define V1SL_PBC_RELEASE_PATH_INFO(_SYS_PTR,_DETAIL_PTR)
```

By calling this macro, the PBC driver firmware returns the path description previously determined with the output macro *V1SL_PBC_GET_PATH_INFO()* to the system environment in the form of two pointers (system pointer and system detail pointer).

Output Macro:	**V1SL_PBC_RELEASE_PATH_INFO**	
Meaning:	Release path information for the PBC driver	
Transfer:		
Parameter	Type, Attribute/ Value Range	Meaning
_SYS_PTR	V1SL_SYS_SYSTEM_PTR	System pointer
_DETAIL_PTR	V1SL_SYS_PBC_DETAIL_PTR	Detail pointer of the PBC driver firmware; the V1SL fills in this data

Recommended actions:
•
Corresponding input functions:

18.6 Interface Expansion of the Output Macros for Multi Instance Operation

When recognizing multi instance operation, the following output macros receive an additional relevant parameter *_HANDLE*. It is used for referencing the communication channel in the level above (component which was called (C0 or C2 firmware)).

The presentation of the macro prototypes used below is to clarify only the differences in the call parameters in comparison to the non-multi instance variant of the macro. For this reason, the prototype presentation is interrupted with '...', and thus not specified completely.

Prototype:

```
#define PBC_C0C1_EVENT_INDICATION(...,_HANDLE)
#define PBC_C2_EVENT_INDICATION(...,_HANDLE)
```

Output Macro:	**PBC_C0C1_EVENT_INDICATION PBC_C2_EVENT_INDICATION**	
Meaning:	Services of the PBC driver to the firmware level above	
Transfer:		
Parameter	Type, Attribute/Value Range	Meaning
...		Parameters that depend on the concrete function
_HANDLE	Unsigned8	Handle of the firmware level above
Corresponding input functions		
v1sl_c0c1_perform_services(), v1sl_c2_perform_services()		

19 Attributes and Data Types

19.1 General

Memory attributes are intended for optimally fitting the V1SL firmware package in the corresponding environment regarding an individual memory model. Optimum setting is decisive for the generated program- and data memory size as well as for program runtimes. If the attributes are not defined, the compiler used automatically sets the attributes to the selected basic memory model (for example, small, medium, large).

19.2 Program Memory Attributes of the PBC Driver

Attribute Name	Description
PBC_IFA_CODE_ATTR	Program memory attribute of the interface functions between C0 or C2 firmware and PBC driver
PBC_SYS_CODE_ATTR	Program memory attribute of the PBC driver interface functions to the system environment; PBC driver output macros to the system
PBC_INT_CODE_ATTR	Internal program memory attribute of PBC driver

19.3 Data Memory Attributes of the PBC Driver

Attribute Name	Description
PBC_INT_DATA_ATTR	Internal data memory attribute PBC driver

19.4 Basic Pointer Types

Data Type Name	Description
PBC_INT_VOID_PTR	Pointer to a void data with *PBC_INT_DATA_ATTR*
PBC_INT_UNSIGNED8_PTR	Pointer to an Unsigned8 data with *PBC_INT_DATA_ATTR*
PBC_INT_UNSIGNED16_PTR	Pointer to an Unsigned16 data with *PBC_INT_DATA_ATTR*
PBC_INT_UNSIGNED32_PTR	Pointer to an Unsigned32 data with *PBC_INT_DATA_ATTR*

19.5 Detail Info Structure of the PBC Driver

V1SL_STRUC_PBC_DETAIL (pbc_open_device()/V1SL_PBC_GET_PATH_INFO())		
Parameter	**Type/Value**	**Description**
device_type	Unsigned8 / PBC_DEVICE_TYPE_DPC31	PBC Type: • DPC31
dev_installed	Unsigned8 / Prior to calling *pbc_open_device()*, assign PBC_DEVICE_TYPE_UNUSED; otherwise ignore	Internal flag of the PBC driver
baud_control	Unsigned8 / 000 001..255	Root value of the monitoring time for each baudrate found (refer to Section 16.2) • After the PBC driver has found a baudrate, it sets a sufficient monitoring time dependent on the baudrate • The user uses the presetting of the PBC driver; that means, after a baudrate was found, the value that was set by the user is

		converted to a monitoring time independent of the baudrate, according to the following formula: $T_{baud\_control}$ = 10ms*baud_control^2
mintsdr	Unsigned16 / 000..010 0011..0255	Time slave waits after the end of a received telegram until it starts sending the response (in tBit, refer to Section 16.4 • After finding a baudrate, the slave always sets mintsdr = 11 (only C0 is generated) • After finding a baudrate, the slave sets a value that depends on the baudrate (C0 and C2 firmware is generated) • After finding a baudrate, the slave always sets this value which was specified by the user
user_wd_value	Unsigned16 / 00000 00001..65535	Value of user watchdog (no time value!; refer to Section 16. • Switching off the user watchdog • Value of user watchdog until expiration
		Sub-Structure dpc31
Parameter	**Type/Value**	**Description**
com_asic_mem_address	DPC31_LL_PTR	Memory address of the PBC DPC31. This value has to be filled in only if the configuration switch V1SL_CFG_ENVIRONMENT_DPC31 is selected (refer to Section 25.1
com_mode	Unsigned16 / OR operation of the following possible values (refer to Section 20.1 'PBC Parameter' on page 153): PBC_DPC31_MODE_ DIS_START_CONTROL PBC_DPC31_MODE_ EOI_TIMEBASE_1u PBC_DPC31_MODE_ EOI_TIMEBASE_1m PBC_DPC31_MODE_ EARLY_RDY	DPC31 hardware settings: • Monitoring of the subsequent startbit in asynchronous physics, valid for the pure operation of the C2 firmware • PBC interrupt-free time (minimum time between two PBC interrupts) of 1µs • PBC interrupt-free time (minimum time between two PBC interrupts) of 1ms • Early ready signal
com_mode_syn_low	Unsigned16 / PBC_DPC31_MODE_SYN_L_ ADD_SIGNAL PBC_DPC31_MODE_SYN_L_ GIM_ENABLE PBC_DPC31_MODE_SYN_L_ QUICK_SYNC	DPC31 hardware settings for synchronous bus physics: • not tested yet • not tested yet • not tested yet
com_mode_syn_high	Unsigned16 /	DPC31 hardware settings for synchronous bus physics:

	PBC_DPC31_MODE_SYN_H_CLOCK_IN_2_MHZ PBC_DPC31_MODE_SYN_H_CLOCK_IN_4_MHZ PBC_DPC31_MODE_SYN_H_CLOCK_IN_8_MHZ PBC_DPC31_MODE_SYN_H_CLOCK_IN_16_MHZ PBC_DPC31_MODE_SYN_H_BAUD_ALL PBC_DPC31_MODE_SYN_H_PREAMBLE_1_BYTE PBC_DPC31_MODE_SYN_H_PREAMBLE_2_BYTE PBC_DPC31_MODE_SYN_H_PREAMBLE_4_BYTE PBC_DPC31_MODE_SYN_H_PREAMBLE_8_BYTE	• not tested yet • not tested yet • not tested yet • not tested yet • not tested yet • not tested yet • not tested yet • not tested yet • not tested yet
com_syn_physic	Unsigned8 / FALSE TRUE	Utilization of DPC31 support for synchronous bus physics: • don't use • use; in this case, the settings of the elements *com_mode_syn_low* and *com_mode_syn_high* are accepted; not tested yet
com_mac_address	Unsigned8 / 000 001..125 126 127..255	PROFIBUS station address of the slave • impermissible value • permissible value • Value intended exclusively for the functionality of node initialization (refer to Section 10.2.2.10 • impermissible area
		• • •
com_user_ram_segments	Unsigned8 / 000 001.. DPC31_USER_RAM_SEGMENTS DPC31_USER_RAM_SEGMENTS+1..255	Number of internal DPC31 RAM segments (32 bytes each) that the PBC driver may use as a maximum: • impermissible value • permissible value • impermissible value
c0_sub_funct	Unsigned8 / OR operation of the following three values V1SL_SUB_FUNCT_C0_NO_PUBLISHER V1SL_SUB_FUNCT_C0_NO_ADD_CHANGE V1SL_SUB_FUNCT_AL_ALARM_SAP	 Deactivates publisher functionality Deactivates SAP 55, which is used to receive 'Set Slave Address' telegrams Activates SAP 50 to be utilized for alarm hanlign in addition to SAP 51.
c0_dx_tact_beat_out	Unsigned8 / OR operation of the following values	Activates hardware signals controlled via the bus

	PBC_DPC31_DX_OUT PBC_DPC31_TACT_BEAT_OUT	• Not set: C31 core of DPC31 is in control of port PB_3 • Set: Port PB_3 is controlled by receiving new outpuut data • Not set: C31 core of DPC31 is in control of port PB_2 • Set: Port PB_2 is controlled by receiving a special Global_Control telegram; in addition, the parameter c0_tact_group must be used
c0_tact_group	Unsigned8 /	Group mask for the Global_Control telegram, which signals the start of a bus cycle, when the master is operated in Equidistant Mode
c0_ssa_buf_len	Unsigned8 / 000 001..003 004..244 245..255	Length of slave address data buffer (refer to Section 10.2.2.10): • Deactivates 'Set Slave Address' functionality • impermissible length • permissible length • impermissible length
c0_prm_buf_len	Unsigned8 / 008..244 245..255	Length of the parameterization data buffer (refer to Section 10.2.2.11 • permissible length • impermissible length
c0_cfg_buf_len	Unsigned8 / 000 001..244 245..255	Length of the expected configuration/configuration sent by the user buffers (refer to Section 10.2.2.12 • impermissible length • permissible length • impermissible length
c0_inp_buf_len	Unsigned8 / 000..244 245..255	Length of the input data buffer: • permissible length • impermissible length
c0_outp_buf_len	Unsigned8 / 000..244 245..255	Length of the output data buffer • permissible length • impermissible length
c0_diag_buf_len	Unsigned8 / 000..238 239..255	Length of the user diagnostic buffer (6 bytes standard diagnosis is not part of the length, refer to Section 10.1.2.12 • permissible length • impermissible length
sc_filter_table_len	Unsigned8 / 000	Maximum length of the table, which defines the filters used for the subscriber functionality (what data from which publisher are relevant for operation) • subscriber not supported
c1_pdu_size	Unsigned8 / 000 001..003	Size of data buffer at the 'Server SAP' (SAP 51, refer to Section 10.2.4): • acyclic services (AL, C1) not activated; paramters starting with *c1* are *not* relevant • impermissible length

	004..244 245..255	• permissible length; acyclic services (AL, C1) activated; paramters starting with *C1* are relevant • impermissible length
		• •
c2_pdu_size	Unsigned8 / 000 001.047 001.063 048..244 064..244 245..255	Size of a data PDU buffer for C2 connections: • Acyclic C2 communication is disabled; parameters starting with *c2* are *not* relevant • impermissible length (when message is *not* distributed any further) • impermissible length (when message is distributed any further) • permissible length (when message is not distributed any further); acyclic C2 communication is enabled; parameters starting with *c2* are relevant • permissible length (when message is distributed any further); acyclic C2 communication is enabled; parameters starting with *c2* are relevant • impermissible length
c2_connect_count	Unsigned8 / 000 001..007 008..255	Maximum number of C2 connections (the PBC driver utilizes SAPs starting with number 48 downward): • impermissible number • permissible number • impermissible number
		• • •
tm_pdu_size	Unsigned8 / 000	Receiving of time stamp telegrams is supported • Not supported

20 Encoding Rules

20.1 PBC Parameters

		pbc_open_device()
		device_type
Symbolic Value	Numeric Value	Description
PBC_DEVICE_TYPE_UNUSED	0xFF	PBC unused
PBC_DEVICE_TYPE_DPC31	0x03	PBC DPC31
		com_mode
Symbolic Value	Numeric Value	Description
PBC_DPC31_MODE_ DIS_START_CONTROL	0x0001	• Monitoring the following startbit in asynchronous physics; valid for pure C2 firmware operation
PBC_DPC31_MODE_ EOI_TIMEBASE_1u	0x0000	• PBC interrupt-free time (minimum time between two PBC interrupts) of 1µs
PBC_DPC31_MODE_ EOI_TIMEBASE_1m	0x0020	• PBC interrupt-free time (minimum time between two PBC interrupts) of 1ms
PBC_DPC31_MODE_ EARLY_RDY	0x0040	• Early ready signal
		c0_dx_tact_beat_out (DPC31)
Symbolic Value	Numeric Value	Description
PBC_DPC31_DX_OUT	0x40	• Port PB_3 is controlled by receiving new outpuut data
PBC_DPC31_TACT_BEAT_OUT	0x20	• Port PB_2 is controlled by receiving a special Global_Control telegram
		user_ram_segments (DPC31)
Symbolic Value	Numeric Value	Description
DPC31_USER_RAM_SEGMENTS	001..173	• Number of 32 byte segments within the communication RAM available for V1SL; remaining part is available to user (if sufficient for V1SL)
		com_c0c1c2_support (DPC31)
Symbolic Value	Numeric Value	Description
DPC31_C0_SUPPORT	0x01	• Activates cyclical services (C0)
DPC31_C1_SUPPORT	0x02	• Activates acyclical services with the parameterization master (AL, C1)
DPC31_C2_SUPPORT	0x20	• Activates acyclical services with other masters (C2)

20.2 Baudrates

		pbc_get_baudrate()
Symbolic Value	Numeric Value	Description
PBC_BAUDRATE_12M	0x00	• Baudrate 12Mbaud
PBC_BAUDRATE_6M	0x01	• Baudrate 6Mbaud
PBC_BAUDRATE_3M	0x02	• Baudrate 3Mbaud
PBC_BAUDRATE_1_5M	0x03	• Baudrate 1.5Mbaud
PBC_BAUDRATE_500k	0x04	• Baudrate 500kBaud
PBC_BAUDRATE_187_5k	0x05	• Baudrate 187.5kBaud
PBC_BAUDRATE_93_75k	0x06	• Baudrate 93.75Baud
PBC_BAUDRATE_45_45k	0x07	• Baudrate 45.45kBaud

PBC_BAUDRATE_19_2k	0x08	• Baudrate 19.2kBaud
PBC_BAUDRATE_9_6k	0x09	• Baudrate 9.6kBaud
PBC_BAUDRATE_INVALID	0xFF	• Baudrate search deactivated, or no baudrate found/present at this time

20.3 Watchdog States

		pbc_get_wd_state()
Symbolic Value	Numeric Value	Description
PBC_WD_STATE_BAUD_SEARCH	0x00	• Baudrate search active
PBC_WD_STATE_BAUD_CONTROL	0x40	• Baudrate search monitoring
PBC_WD_STATE_DP_MODE	0x80	• DP watchdog mode
PBC_WD_STATE_OFF	0xC0	• No baudrate search active

21 Ressources

21.1 General

The tables below provide component-granular information about the required program and data memory of the PBC driver firmware. The requirement depends on the following:

- the tool set used
- the selection of single or multi device operation
- the assignment of the system output macros
- the selection of the memory attributes
- on additional configuration switches (for example, static or dynamic memory management)

For this reason, only basic values (minimum/maximum) are specified for the memory requirement.

The user only takes the memory of the components into consideration he wants to generate (refer to Section 25.1). The stated data memory requirement identifies the amount that is needed per utilized PBC.

In addition, further required resources are specified.

Note: To determine the program and data memory requirement as well as the requirement for timers for the complete firmware, the values in Section 13'

21.2 System Interface

These memory shares are to be included in the overall memory requirement for each V1SL generation.

	System Interface PBC Driver
Keil C51	...
Tasking C166	>2,2 (near)
Borland C++	...
MS Visual C++	...

Table 21: Program Memory Requirement System Interface PBC Driver (in KBytes)

	SystemInterface PBC Driver
Keil C51	...
Tasking C166	0
Borland C++	...
MS Visual C++	...

Table 22: Data Memory Requirement System Interface PBC Driver (in Bytes)

21.3 DPC31 Driver

	C0	AL/C1/C2
Keil C51	...	...
Tasking C166	...	...
Borland C++	...	...
MS Visual C++	...	...

Table 23: Program Memory Requirement DPC31 Driver (in KBytes)

	C0	AL/C1/C2
Keil C51	...	...
Tasking C166	...	...
Borland C++	...	...
MS Visual C++	...	...

Table 24: Data memory Requirement DPC31 Driver (in Bytes)

In addition to the memory specified in Table 24, an internal DPC31 data memory is necessary per PBC for receive and send buffers, which is limited to a 6Kbyes maximum.

22 Delivery Package

22.1 V1SL Archive File

22.2 Format of the Source Files

The source files were designed in a way that the firmware can be generated on the following platforms:

- DOS
- Windows®/Win95®/WinNT®
- Unix®

The following points were taken into account:

- For file names, only lower case letters were used.
- To specify paths within source files and header files, the Unix® style in the form of '/' (slash) was used as separator for directory and file names.
- Source files use the EOL-ID in the Unix® style '0x0A'.

22.3 C-Compilers Used

In principle, there are no restrictions regarding the C-compiler types used for generating V1SL firmware. So far, the V1SL has been compiled with the following C(++) compilers or tools:

- Keil C51 V5.x..
- Tasking C166 V...
- Borland C V...
- Microsoft Visual C++ V...
- Watcom C V...
- HiC V...
- CAD-UL...
- PC-Lint V...

23 Directory and File Structure

23.1 General

The directory structure used has proven to be suitable for a large number of other applications, and was accepted for that reason.

23.2 Content of the Firmware Archive

23.2.1 Content of the Main Directory

Directory /Sub-Directory	File Name	Explanation
src_dir/		Directory with source files: • See below
tool_dir/		Directory with tools:

23.2.2 Content of the Source Directory

Directory/ Subdirectory	File Name	Explanation
comm_dev/	v1sl_cfg.txt v1sl_cfg.h	Directory for the global configuration files of all firmware components of a module: • Global Configuration File of V1SL (empty) • Example
common/	v1sl_com.h v1slplau.h	Directory for the export interface declaration of all firmware components of a module: • Export interface of V1SL • Validation of the V1SL configuration settings
v1sl/	v1sl_inc.h	Directory with V1SL sources: • V1SL configuration file to be adapted locally
com_h/	v1sl_pra.h pbc_pra.h c0_com.h c2_com.h pbc_com.h v1sl_int.h v1sllist.h	
com_s/	v1sl_ifa.c v1sl_ifa.h	
c0/	c0_dat.c c0_dat.h c0.c c0.h c0_al.c c0_al.h c0_c1.c c0_c1.h	
c2/		

	c2_dat.c c2_ifa.c c2_pbc.c c2_int.c c2_int.h	
pbc-1/	pbc_dat.c pbc_dat.h pbc_ifa.c pbc_loc.h dpc31ifa.c dpc31cbf.c dpc31int.c dpc31loc.h	

23.2.3 Content of the Application Example

Directory /Sub-Directory	File Name	Explanation
comm_dev/	sys_cfg.h usr_cfg.h	Directory with global configuration data for all firmware components used by the hardware: • System settings • User settings
common/	sys_com.h usr_com.h	Directory with interface descritions for all firmware components used by the hardware: • System settings • User settings
system/	sys_51.asm sys_main.c sys_inc.h	Program start: • Start routine • Start-up and settings • Settings
usr/	usr.c usr.h usr_inc.h	Main program: • Main program • User settings • Settings

24 Configuration

24.1 Filling in the 'v1sl_cfg.h' File by the User

This is the global configuration file of the V1SL that is located in the directory 'v1sl/src_dir/comm_dev/'. In the delivery package of the V1SL, this file is included as a text file ('v1sl_cfg.txt').

Note: Since header files are generally designated with '*.h', the user has to rename the file 'v1sl_cfg.txt' to 'v1sl_cfg.h' himself. This ensures that if a new V1SL version is transferred to the user's directory structure, the previous configuration is not overwritten.

Category Switches	Fill-In Requirement[2]	Value Range	Explanation
General			
Generation Tool			The settings in this section cause the V1SL to respond to specifics of different tool sets at generation (only one tool set from the selection below is to be defined):
V1SL_TOOL_CHAIN_TASKING_51	>>> m <<<		• C51 tool set by Tasking
V1SL_TOOL_CHAIN_TASKING_166			• C166 tool set by Tasking
V1SL_TOOL_CHAIN_KEIL_51			• C51 tool set by Keil
V1SL_TOOL_CHAIN_KEIL_166			• C166 tool set by Keil
V1SL_TOOL_CHAIN_BORLAND			• Borland C(++)
V1SL_TOOL_CHAIN_MICROSOFT			• Microsoft C / Visual C(++)
Firmware Sub-Components of V1SL			The settings in this section specify which V1SL firmware components are to be generated:
V1SL_CFG_COMPONENT_C0 (**C0**)	>>> o <<<	0≤x≤255	Not defined, or defined and value = 0: • The C0 firmware is not generated. Defined and value > 0: • The state machine of the cyclical services of the V1SL (C0, MSCY1S) is generated. • The value determines the number of the instances supported as a maximum. • If the value is > 1 (multi-instance), the prototypes for all C0- and C2 firmware user interface functions are provided with a handle (*Unsigned8 handle*) as the last parameter. In this case, the handle parameter (*_HANDLE*) of the C0/C2 output macros to the user also contains a valid value. • The following five configuration

[2] This column indicates in a compressed form, whether and under which conditions a macro has to be filled. The format follows a syntax similar to C. The operators are short forms of generating switches that are listed in bold type in parantheses in the 1st column (e.g. (**C0**)). The result of each expression is listed behind a colon. The meaning of the results is as follows: >>>m<<< (mandatory; must be filled under the specified preconditions); >>>o<<< (optional; under the specified preconditions, it is up to the user to fill the macro). If a condition does not apply, the user does not have to fill the macro.

			switches are effective.
V1SL_CFG_COMPONENT_SUB_AL (**SUB_AL**)	if **C0>0**: >>> o <<<		Not defined: • The alarm state machine of V1SL (AL, MSAL1S) is not generated. Defined: • The alarm state machine of V1SL (AL, MSAL1S) is generated.
V1SL_CFG_COMPONENT_SUB_C1 (**SUB_C1**)	if **C0>0**: >>> o <<<		Not defined: • The state machine of the acyclical C1 services of V1SL (C1, MSAC1S) is not generated. Defined: • The state machine of the acyclical C1 services of V1SL (C1, MSAC1S) is generated.
V1SL_CFG_COMPONENT_C2 (**C2**)	>>> o <<<	0≤x≤255	Not defined, or defined and value = 0: • The C2 firmware is not generated. Defined and value > 0: • The state machine of the acyclical C2 services of V1SL (C2, MSAC2S) is generated. • The value determines the number of instances supported as a maximum. • If the value is > 1 (multi-instance), the prototypes for all C0- and C2 firmware user interface functions are provided with a handle (*Unsigned8 handle*) as last parameter. In this case, the handle parameter (*_HANDLE*) of the C0/C2 output macros to the user contains also a valid value. • The following five configuration switches are effective.
V1SL_CFG_C2_CONNECTION_NUMBER_MAX	if **C2>0**: >>> m <<<	1≤x≤49	• Specifies the maximum number of connections of the C2 firmware component. This value is compared with the indication of a number of connections in the C2 detail structure (*V1SL_STRUC_C2_DETAIL*) when a C2 communication channel (*v1sl_c2_open_channel()*) is opened.

V1SL_CFG_C2_SUBNET	if **C2>0**:
	
	>>> m <<<
		1
		
		2
			• C2 firmware is integrated in a master or slave module that is directly connected to the DP bus (corresponds to the declaration *V1SL_SUBNET_LOCAL* in the export interface file 'v1sl_com.h').
			• C2 firmware is integrated in a module that is connected to the DP bus via an IM/Link (corresponds to the declaration *V1SL_SUBNET_REMOTE* in the export interface file 'v1sl_com.h').

V1SL_CFG_C2_DEBUG_ENABLE (**C2_DEB**)	if **C2>0**: >>> o <<<		Not defined: • The communication sequence is not recorded. Defined: • For the C2 services, a trace buffer (orgaanized as a ring buffer) is set up. Note: this configuration switch should not be activated during normal

			operation, but only during the startup of the firmware; and if errors occur (in consultation with the developer!)
V1SL_CFG_C2_DEBUG_ELEMENT_NUMBER	if **C2>0** and **C2_DEB**: >>> m <<<	1≤x	• Number of elements of the trace buffer: refer to note under *V1SL_CFG_C2_DEBUG_ENABLE*
V1SL_CFG_COMPONENT_DPC31 (**DPC31**)	>>> o <<<	0≤x≤255	Not defined, or defined and value = 0: • the DPC31 PBC driver firmware is not generated. Defined and value is > 0: • the DPC31 PBC driver firmware is generated. • The value determines the number of DPC31 PBCs supported as a maximum. • If the sum of the values for PBCs of all types is > 1, the prototypes for all PBC interface functions are provided with a PBC handle (*Unsigned8 device_handle*) as last parameter.
Characteristics of the system environment in which the V1SL is operated			Characteristics of the system environment of V1SL:
V1SL_CFG_ENVIRONMENT_CONTINUE_ON_FATAL_ERROR	>>> o <<<		Not defined: • The system environment stops processing of additional firmware modules by calling the output macro *V1SL_FATAL_ERROR()*; the call of the output macro does not return to the caller (V1SL). Defined: • If there is a fatal error in V1SL, the system environment can't immediately stop processing the V1SL firmware; the call of the output macro *V1SL_FATAL_ERROR()* returns to the caller (V1SL). In this case, V1SL internally generates mechanisms which stop further function processing and thus any kinds of sequential error indications after a fatal error occurred. The system can be shut down, but without the release of resources that are assigned to V1SL.
V1SL_CFG_ENVIRONMENT_DYNAMIC_MEM (**DYN_MEM**)	>>> o <<<		Not defined: • The V1SL firmware components utilize the local user memory set up statically. The output macros specified below don't have to be filled in. Defined: • The V1SL firmware components allocate and release the local user memory needed dynamically via the system output macros *V1SL_C0C2_ALLOC_LOCAL_MEM()*, *V1SL_C0C2_FREE_LOCAL_MEM()*, *V1SL_PBC_ALLOC_LOCAL_MEM()*, *V1SL_PBC_FREE_LOCAL_MEM()*.

V1SL_CFG_ENVIRONMENT_DYNAMIC_DPC31	if **DPC31=1**: >>> o <<< if **DPC31>1**: >>> m <<<		Not defined: • Is possible only when using exactly one DPC31. • Recommended when using 8 bit processors (for example, 8051) • In the DPC31, the PBC driver accesses data via the '.' operator.' Defined: • Also possible when using one DPC31, mandatory for several DPC31. • Recommended when using >8 bit processors (for example, 80C166, 80x86, Pentium). • In the DPC31, the PBC driver accesses data via the '→' operator. • The system environment has to transfer the address of each DPC31 to the V1SL via the function *pbc_open_device()* specified in the element '*pbc_detail.dpc31.com_asic_mem_address*' of the detail block.
Basic Memory Attributes			Memory attribute settings depend on the tool set used, and are to be assigned accordingly
V1SL_CODE_ATTR_NEAR	>>> o <<<	Attribute	
V1SL_CODE_ATTR_FAR		Attribute	
V1SL_CODE_ATTR_HUGE		Attribute	
V1SL_DATA_ATTR_NEAR		Attribute	
V1SL_DATA_ATTR_FAR		Attribute	
V1SL_DATA_ATTR_HUGE		Attribute	
Basic Data Type Declaration			
V1SL_CFG_TYPE_DEFINITION_ENABLE	>>> o <<<		Not defined: • The system environment is in charge of the setup and announcement of the basic data types (e.g. *Boolean*, *Unsigned8*, usw.) as declared in DPV1. Defined: • The V1SL is in charge of the setup and announcement of the basic data types (e.g. *Boolean*, *Unsigned8*, usw.) as declared in DPV1
System Interface Parameters			
V1SL_SYS_PATH_TYPE	>>> m <<<	Type	• With this parameter, the system environment itself can specify the type of the transfer parameter *sys_path* of the functions *v1sl_c0_open_channel()* and *v1sl_c2_open_channel()* (for example, *Unsigned8*)
V1SL_SYS_PTR_TYPE	>>> m <<<	Type	• With this, the system environment itself can specify the type of the transfer parameter *SYS_PTR_PTR* for the output macros *V1SL_C0C2_GET_PATH_INFO()* and *V1SL_PBC_GET_PATH_INFO() (for example, Unsigned8*)
V1SL_SYS_PATH_OK	>>> m <<<	x	• Return value if the macro calls *V1SL_C0C2_GET_PATH_INFO()* and

			V1SL_PBC_GET_PATH_INFO() are OK.
General Macros to the System			Refer to Section 9.1.2
V1SL_ENTER	>>> o <<<		• Enter uninterruptible segment of firmware processing
V1SL_EXIT			• Exit uninterruptible segment in firmware processing
V1SL_FATAL_ERROR	>>> o <<<		• Indicate fatal error in V1SL to system environment
C0/C2 Macros to the System			Refer to section 9.2.3
V1SL_C0C2_ALLOC_LOCAL_MEM	if		• Allocate internal memory of C0/C2 firmware
V1SL_C0C2_FREE_LOCAL_MEM	**DYN_MEM**: >>> m <<<		• Release internal memory of C0/C2 firmware
V1SL_C0C2_GET_PATH_INFO	if		• Request C0/C2 firmware path information
V1SL_C0C2_RELEASE_PATH_INFO	**C0>0** oder **C2>0**: >>> m <<<		• Release C0/C2 firmware path information
PBC Driver Macros to the System			Refer to Section 18
V1SL_PBC_ALLOC_LOCAL_MEM	if		• Allocate internal memory of PBC firmware
V1SL_PBC_FREE_LOCAL_MEM	**DYN_MEM**: >>> m <<<		• Release internal work memory of PBC firmware
V1SL_PBC_GET_PATH_INFO	If		• Request PBC firmware path information
V1SL_PBC_RELEASE_PATH_INFO	**C0>0** or **C2>0**: >>> m <<<		• Release PBC firmware path information
Data Memory Fill/Copy Macros			These macros are used to design the data memory fill/copy activities as effectively as possible, according to the V1SL environment. This should be done by taking the data memory attributes into consideration. If the user does not declare this, the V1SL presets the macros with standard settings.
v1sl_fill_byte_ifa__	>>> o <<<		• Fill one byte field of the data attribute *V1SL_IFA_DATA_ATTR*
v1sl_fill_byte_int__			• Fill one byte field of the attribute *V1SL_INT_DATA_ATTR*
v1sl_fill_byte_ll__			• Fill one byte field of the data attribute *V1SL_LL_DATA_ATTR*
v1sl_copy_byte_from_ifa_to_ifa__			• Copy a byte field of the data attribute *V1SL_IFA_DATA_ATTR* to a byte field of the data attribute *V1SL_IFA_DATA_ATTR*
v1sl_copy_byte_from_int_to_int__			• Copy a byte field of the data attribute *V1SL_INT_DATA_ATTR* to a byte field of the data attribute *V1SL_INT_DATA_ATTR*
v1sl_copy_byte_from_ll_to_ll__			• Copy a byte field of the data attribute *V1SL_LL_DATA_ATTR* to a byte field of the data attribute *V1SL_LL_DATA_ATTR*
v1sl_copy_byte_from_ifa_to_int			• Copy a byte field of the data attribute

—			*V1SL_IFA_DATA_ATTR* to a byte field of a data attribute *V1SL_INT_DATA_ATTR*
v1sl_copy_byte_from_int_to_ifa —			• Copy a byte field of the data attribute *V1SL_INT_DATA_ATTR* to a byte field of the data attribute *V1SL_IFA_DATA_ATTR*
v1sl_copy_byte_from_ifa_to_ll_ –			• Copy a byte field of the data attribute *V1SL_IFA_DATA_ATTR* to a byte field of the data attribute *V1SL_LL_DATA_ATTR*
v1sl_copy_byte_from_ll_to_ifa_ –			• Copy a byte field of the data attribute *V1SL_LL_DATA_ATTR* to a byte field of the data attribute *V1SL_IFA_DATA_ATTR*
v1sl_copy_byte_from_int_to_ll_ –			• Copy a byte field of the data attribute *V1SL_INT_DATA_ATTR* to a byte field of the data attribute *V1SL_LL_DATA_ATTR*
v1sl_copy_byte_from_ll_to_int_ –			• Copy a byte field of the data attribute *V1SL_LL_DATA_ATTR* to a byte field of the data attribute *V1SL_INT_DATA_ATTR*
V1SL			
Program Memory Attributes			Refer to Figure 1
V1SL_IFA_CODE_ATTR	>>> o <<<		• (1)
V1SL_SYS_CODE_ATTR			• (2)
V1SL_INT_CODE_ATTR			• (5)
Data Memory Attributes			Refer to Figure 1
V1SL_IFA_DATA_ATTR	>>> o <<<		• [1]
V1SL_LL_DATA_ATTR			• [3]
V1SL_SYS_DATA_ATTR			• [2]
V1SL_INT_DATA_ATTR			• [4]
C0 User Identification			
V1SL_C0_USER_ID_TYPE	if **C0>0**: >>> m <<<	x	• The C0 firmware user can specify the type of the transfer parameter *user_id* of the C0 firmware functions by itself (e.g. Unsigned8).
C2 User Identification			
C0 Output Macros to User			Refer to <<10.2.2
V1SL_C0_OPEN_CHANNEL_DONE	if **C0>0** and not **C0_RQB**: >>> o <<<		• Completion of user request *v1sl_c0_open_channel()*
V1SL_C0_CLOSE_CHANNEL_DONE			• Completion of user request *v1sl_c0_close_channel()*
V1SL_C0_WITHDRAW_DONE			• Completion of user request *v1sl_c0_withdraw()*
V1SL_C0_DP_WD_TIMEOUT			• Indicates a DP watchdog timeout event
V1SL_C0_WD_STATE_REPORT			• Indicates a DP watchdog state change
V1SL_C0_DP_STATE_REPORT			• Indicates a DP state
V1SL_C0_LED_STATE_REPORT			• Indicates the bus error LED state
V1SL_C0_DATA_EXCHANGE_ACTIVE			• Indicates start of data exchange between parameterization master and slave.
V1SL_C0_REAL_CFG_BUFFER_ CHANGED			• Indicates that a free configuration buffer can be fetched by the user.

V1SL_C0_DIAG_CHANGED			• Completion of the user request *v1sl_c0_set_diag()*
V1SL_C0_DIAG_FETCHED			• Indicates that a diagnosis set by the user with *v1sl_c0_set_diag()* was fetched.
V1SL_C0_NEW_SSA			• Indicates receipt of Set Slave Address data
V1SL_C0_NEW_PRM			• Indicates receipt of parameterization data
V1SL_C0_NEW_CFG			• Indicates receipt of configuration data sent by the master
V1SL_C0_CLEAR			• Indicates 'Clear'/'Unclear' mode
V1SL_C0_SYNC			• Indicates the 'Sync'/'Unsync' mode
V1SL_C0_FREEZE			• Indicates the 'Freeze'/'Unfreeze' mode
AL Output Macros to User			Refer to Section 10.2.3
V1SL_AL_STATE_REPORT	If		• Indicates a state change of the alarm state machine
V1SL_AL_ALARM_ACK	**C0>0** and not **C0_RQB** and **SUB_AL**: >>> o <<<		• Completion of user request *v1sl_al_set_alarm()*
C1 Output Macros to User			Refer to Section 10.2.4
V1SL_C1_READ_DS	if		• Indicates request 'Read Data Set', by parameterization master.
V1SL_C1_WRITE_DS	**C0>0** and not **C0_RQB** and **SUB_C1**: >>> o <<<		• Indicates request 'Write Data Set', by parameterization master.
C2 Output Macros to User			Refer to Section <<10.2.4
V1SL_C2_OPEN_CHANNEL_DONE	if		• Completion of user request *v1sl_c2_open_channel()*
V1SL_C2_CLOSE_CHANNEL_DONE	**C2>0**		• Completion of user request *v1sl_c2_close_channel()*
V1SL_C2_INITIATE	and not		• Indicates C2 master request 'Establish Connection'
V1SL_C2_ABORT	**C2_RQB**:		• Indicates 'Shut Down Connection'
V1SL_C2_USER_ABORT_DONE	>>> o <<<		• Completion of user request *v1sl_c2_user_abort()*
V1SL_C2_DATA_TRANSPORT			• Indicates C2 master request 'Data Transport'
V1SL_C2_READ_DS			• Indicates C2 master request 'Read Data Set'
V1SL_C2_WRITE_DS			• Indicates C2 master request 'Write Data Set'
PBC Driver			
Program Memory Attributes			Refer to Figure 1
PBC_IFA_CODE_ATTR	>>> o <<<		• (4)
PBC_SYS_CODE_ATTR			• (3)
PBC_INT_CODE_ATTR			• (6)
Data Memory Attributes			Refer to Figure 1
PBC_INT_DATA_ATTR	>>> o <<<		• [5]
Context change between PBC driver and C0/C2 firmware			Refer to Section 18
PBC_C0C1_EVENT_INDICAT	if		• Indicates PBC driver events to the C0

ION	**C0>0**: >>> m <<<		firmware
PBC_C2_EVENT_INDICATION	if **C2>0**: >>> m <<<		• Indicates PBC driver events to the C2 firmware
Hardware Features of the DPC31			
PBC_DPC31_SPEC_HW_MODE	if **DPC31>0**: >>> o <<<		Not defined: • This is the presetting when the address line A_0 of the DPC31 is wired as the least significant address line. Defined • Special HW mode where the address line A_1 is the least significant address line. The result is that each byte (8 bits) in DPC31 is one word (16 bits) wide. In this case, the PBC driver firmware is to process only the less significant 8 bits of each of these words. In addition, the PBC driver copies all data of the telegram buffers to a temporary storage which is transferred to the user. This is to ensure that the behavior of the data buffers is uniform according to the specification at the interface to the user. This functionality is not yet implemented!!!

The figure below shows the program memory attributes and data attributes that are to be set, and their relationship with the firmware components of a module.

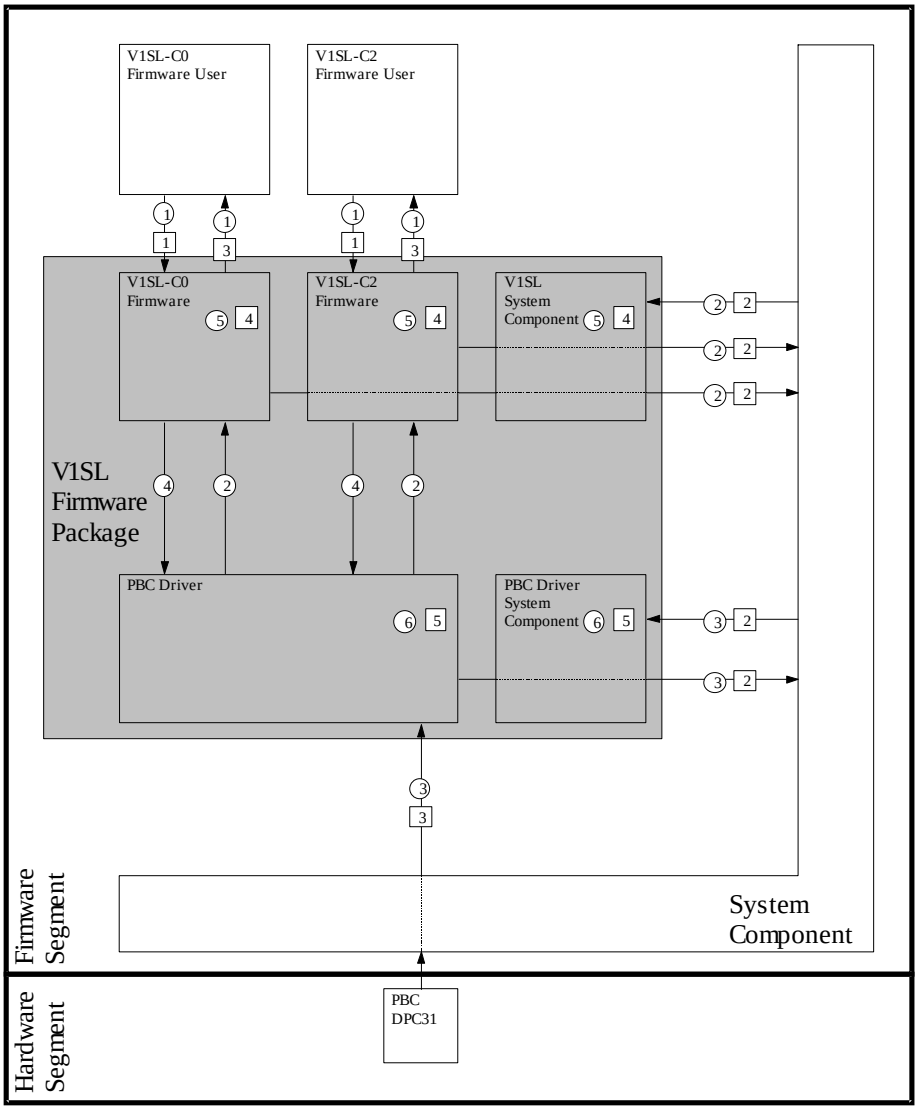


Figure 1: Program Attributes and Data Memory Attributes

24.2 User Fills in the File 'v1sl_inc.h'

This is a local configuration file of the V1SL that is located in the directory 'v1sl/src_dir/v1sl/'. In the V1SL shipping package, this file is included as a text file ('v1sl_inc.txt').

Note: Since header files are generally designated with '*.h', the user has to rename the file 'v1sl_cfg.txt' to 'v1sl_cfg.h' himself. This ensures that if a new V1SL version is transferred to the user's directory structure, the previous configuration is not overwritten.

The file 'v1sl_inc.h' is included into all source files of the V1SL firmware package. With this file all configuration and export interfaces of adjacent firmware components and of V1SL itself are made known to V1SL.

Note: The hierarchy for including each component always has to be → configuration file ('*cfg.h') of the component **before** export interface ('*com.h') of the component!

The table below shows a sample assignment of the local configuration file 'v1sl_inc.h'.

C Command Line	Brief Description
#include "com_typ.h"	Include neutral type declarations that all firmware components can utilize.
#include "sys_cfg.h"	Include configuration file of the system component
#include "sys_com.h"	Include export interface file of the system component
#include "v1sl_cfg.h"	Include configuration file of V1SL
#include "v1sl_com.h"	Include export interface file of V1SL
#include "user_cfg.h"	Include configuration file of the higher level (user, application)
#include "user_com.h"	Include export interface file of the higher level (user, application)

Table 1: Sample Assignment of the Local V1SL Configuration File 'v1sl.inc.h'

25 Generation

Note: The description below refers exclusively to the generation tools for the generation of the sample applications. If others than these are used, the user has to design his generation data himself. In that case, existing dependencies of the source files are provided in the included make files, or they can be specified automatically with corresponding tools.

25.1 Preparation

Prior to starting generation, the tools needed have to be set up in the directory ‘v1sl/tool_dir/’. In general, this includes the following:

- Generation tools: compiler/assembler/linker/ etc.
- Additional generation tools: make, symbol preprocessor, etc.

Then, the following files are to be adapted to individual needs in the directory ‘v1sl/bat_dir/’:

- '_mak51.bat': general settings
- 'v1sl_51.mak': make file for the sample application for the Keil C51 compiler

25.2 Compilation

The firmware package V1SL, in connection with the sample application, is generated by executing the included batch file ‘_gen51.bat’.

25.3 Locating the Memory Units of V1SL

If the C166 tool set of the Tasking company is used, one of the last steps for generating the sample applications is the arrangement of the program- and data elements of the V1SL in the memory of the destination platform (locating). The user can assume the following V1SL memory classes:

- V1SL_DATA_NEAR
- V1SL_DATA_FAR
- V1SL_DATA_HUGE
- V1SL_DATA_SHUGE (only for C166 compiler versions >= V6.0)
- V1SL_CODE
- V1SL_CONS
- PBC_DATA_NEAR
- PBC_DATA_FAR
- PBC_DATA_HUGE
- PBC_DATA_SHUGE (only for C166 compiler versions >= V6.0)
- PBC_CODE
- PBC_CONS

If one or several classes do not exist because of the selection of the memory attributes, there may be warnings during the locating process. The easiest way to remove these warnings is by not locating memory classes that don't exist (delete them from the locating specification).

26 Literature

[1] PROFIBUS-DP Standard, EN50170
Version ...; ... .

[2] PROFIBUS, Technical Guideline PROFIBUS-DP, Extensions to EN50170 (DPV1)
Version 2.0; April 98.

[3] SIMATIC NET - V1SL User Description (current version).

[4] SIMATIC NET – DPC31 Siemens PROFIBUS-DP Controller with C31 Core Hardware Description (current version).

27 Explanation of Terms

AKF	General ID format for slave configuration data
BF-LED	Bus error LED (in a slave module)
CBF	Call Back Function
DB	Data Block
Diag_Upd_Delay	Parameter of the master parameter set (refer to Item 11 in [2])
DP	Distributed periphery
DPC31	DP Controller with integrated C31 kernel, suitable for slave applications
DPV1	Extension of the PROFIBUS-DP standard (refer to [2]: • Parameterization features • Diagnostic features • Asynchronous communication between parameterization master (C1 master) and slave • Asynchronous communication between C2 master and slave
DS	Data set
FDL	Fieldbus Data Link
FW	Firmware
GC	Global-Control; DP message with control function for all/certain slaves
GSD	Device description data file that describes slave parameters, refer to [2]
HW	Hardware
IM	Interface Module (e.g. Siemens ET200 module IM153)
IMF-LED	Interface Module Error LED (in a slave module)
Min-Slave-Interval	Slave parameter (part of GSD file) which specifies the minimum time interval between two telegrams addressed to the slave.
NRS-PDU	Negative Response Protocol Data Unit (negative response telegram)
PBC	PROFIBUS Controller (ASIC)
PDU	Protocol Data Unit; part of the telegram with structured organization.
PNO	PROFIBUS Trade Organization
PROFIBUS	PROcess FIeld BUS; industrial or fieldbus system
REQ-PDU	Request Protocol Data Unit (request telegram)

RES-PDU	Response Protocol Data Unit (positive response telegram)
SAP	Service Access Point
SKF	Special ID format for slave configuration data
SM	State Machine; implements sequences of a certain protocol
SW	Software

28 Addresses

PROFIBUS Trade Organization
PNO Office
Haid-und-Neu-Strasse 7
76131 Karlsruhe/Germany
Phone: (0721) 9658-590

Technical Contact Persons at the Interface Center in Germany
Siemens AG
I IA SE DE DP3
Gerd Putschky

Mailing Address:
Postfach 2355
90713 Fuerth/Germany

Street Address:
Wuerzburger Strasse 121
90766 Fuerth/Germany

Phone: (0911) 750 2078
Fax: (0911) 750 2100

EMail:
Gerd.Putschky@siemens.com

Technical Contact Persons at the Interface Center in the USA
PROFIBUS Interface Center
One Internet Plazza
PO Box 4991
Johnson City, TN 37602-4991

Fax : (423) - 262 - 2103

Your Partner:
Tel.: (423) - 262 – 2576

E-Mail:
profibus.sea@siemens.com