



vs | verkehrssysteme

vs | plus 10

Release notes

Cyrill Jucker
02. October 2026

TABLE OF CONTENTS

1	FOREWORD	4
2	NEW PROJECT STRUCTURE FOR THE INTEGRATION OF VERSION 10	4
2.1	Project Setup and Merging	4
2.2	Controller Interface	4
2.3	Interface Functions and Integration	4
3	BUILD	6
3.1	Compiler / Language Standard	6
4	FUNCTIONAL INTERFACE EXTENSIONS	6
4.1	OCIT Color Codes	6
4.2	Newly Added Functions	6
4.2.1	Functions of the Signal Groups	6
4.2.1.1	Mapping a digital output to a signal group	6
4.2.2	Digital Output Functions	6
4.2.2.1	Determination of the switch-on and switch-off time of a digital output	6
4.2.2.2	Digital output access inspection	7
4.2.2.3	Determination of the corresponding signal group of a digital output	7
4.2.3	Functions of On-demand Control	7
4.2.3.1	Checking supported OCIT color codes for a signal group	7
4.2.3.2	Support for the deactivation of transitions in on-demand operation	8
4.2.3.3	Checking the logical state of OCIT color codes of a signal group	8
4.2.3.4	Control of signal groups in on-demand operation (open / closed)	9
4.2.4	Functions of the New Public Transport Module	9
4.2.4.1	Reception of public transport request messages	9
4.2.4.2	Sending response messages (AMLi)	10
4.3	Revised Existing Functions	10
4.3.1	Changed Signatures	10
4.3.1.1	Query of intergreen times of the controller	10
4.3.2	Renamed Functions	10
4.3.2.1	Function change: getSignalGroupPartialNode → getSignalGroupPartialNodeId	10
4.3.2.2	Function change: isSignalGroupInOffStateColour → isOnDemandColourActiveForSignalGroup	11
4.3.2.3	Function change: switchSignalGroupOffStateColour → switchSignalGroupOnDemandColour	11
4.3.2.4	Function change: getElapsedOffStateColour → getElapsedOnDemandColour	11
4.4	Removed Functions	12
4.4.1	Unused Functions	12
4.4.2	Relays, Flashers and Digital Outputs	12
4.4.3	Replacement of the Old Public Transport Module	12
4.5	Potentially Necessary Corrections on Controller Site	12
4.5.1	Functions isOpen / isClosed	12
4.5.2	Functions isOpenMin / isClosedMin	12
4.5.3	Functions getElapsedOpen / getElapsedClosed	13
4.5.4	Functions getElapsedOpenMin / getElapsedClosedMin	13
5	MEMORY ALLOCATION EXTENSIONS	13

5.1	Expansion of Memory IDs	13
5.2	Runtime Restrictions	13
5.3	Memory Access Concept and STL Usage	14
5.4	New Memory IDs for the Public Transport Module	14
6	THE NEW PUBLIC TRANSPORT MODULE	14
6.1	Data Structures and Formats	14
6.2	Public Transport Requests	14
6.3	Public Transport Responses	15
6.4	AMLi Records (Extended Serial R09)	15
7	EXPANSION OF PROCESS DATA	15
7.1	Extended New Public Transport Module Data	15
7.2	Replacement of Historical Line-Course IDs with Process Data in the PT module	15
7.3	General Process Data Output for Public Transport Inquiries	16
7.4	Public Transport Process Data (Extended R09 / AMLi)	17
7.5	Digital Public Transport Process Data (Digital R09 / SREM/SSEM)	18
7.6	Additional Public Transport Process Data (CAM – Cooperative Awareness Message)	18
7.7	Additional Public Transport Process Data (SREM – Signal Request Extended Message)	18
7.8	General Public Transport Module Process Data (Type-independent)	19
8	ONGOING ENHANCEMENTS AND CORRECTIONS	20
8.1	Improvements from vs plus 9.1.0 (Patch 6)	20
8.2	Improvements in Version 10.0.0	21

1 FOREWORD

Version 10 will be rolled out in two stages. Version 10.0.0 includes the functionality of 9.1.0 as well as a completely renewed PT-module including SREM/CAM support. In addition, relays and flasher ports have been standardized as digital outputs. The interface between vs | plus and the controller has been expanded accordingly. Next up is version 10.0.1, which mainly extends the vs | plus on-demand control and incorporates adjustments based on experience in traffic engineering use.

The basics for integrating vs | plus (9.1) can be found in the Implementation Manual (v03-00-06). Think of this document as a continuation of the existing manual. Please pay particular attention to the 4.5 requirements for existing functions.

2 NEW PROJECT STRUCTURE FOR THE INTEGRATION OF VERSION 10

2.1 Project Setup and Merging

The supplied ZIP file contains two directories at the top level (see the following illustration of the file structure):

- «Interface» (outlined in green)
- «VsPlus10-<Build No.>» (outlined in dark blue)

To integrate, follow these steps:

1. Create a shared **root directory**.
2. Copy the entire contents of both folders (including all subdirectories) into this root directory.
3. Keep the existing directory structure unchanged.
4. Configure the include paths **at the project level** so that all "include" directories are included. → Goal: All header files must be resolved correctly during the build.

The **light blue outlined area** contains the vs | plus core.

2.2 Controller Interface

The entire code relevant to the controller is located in the **"Interface" area outlined in green**:

- **Orange outline:**
Declarations of data types and structures for communication between controller and vs | plus
- **Outlined in red:**
 - Configuration data (controller specific)
 - Functional interface declarations

Important: The configuration data must be adapted and supplemented according to the targeted controller.

2.3 Interface Functions and Integration

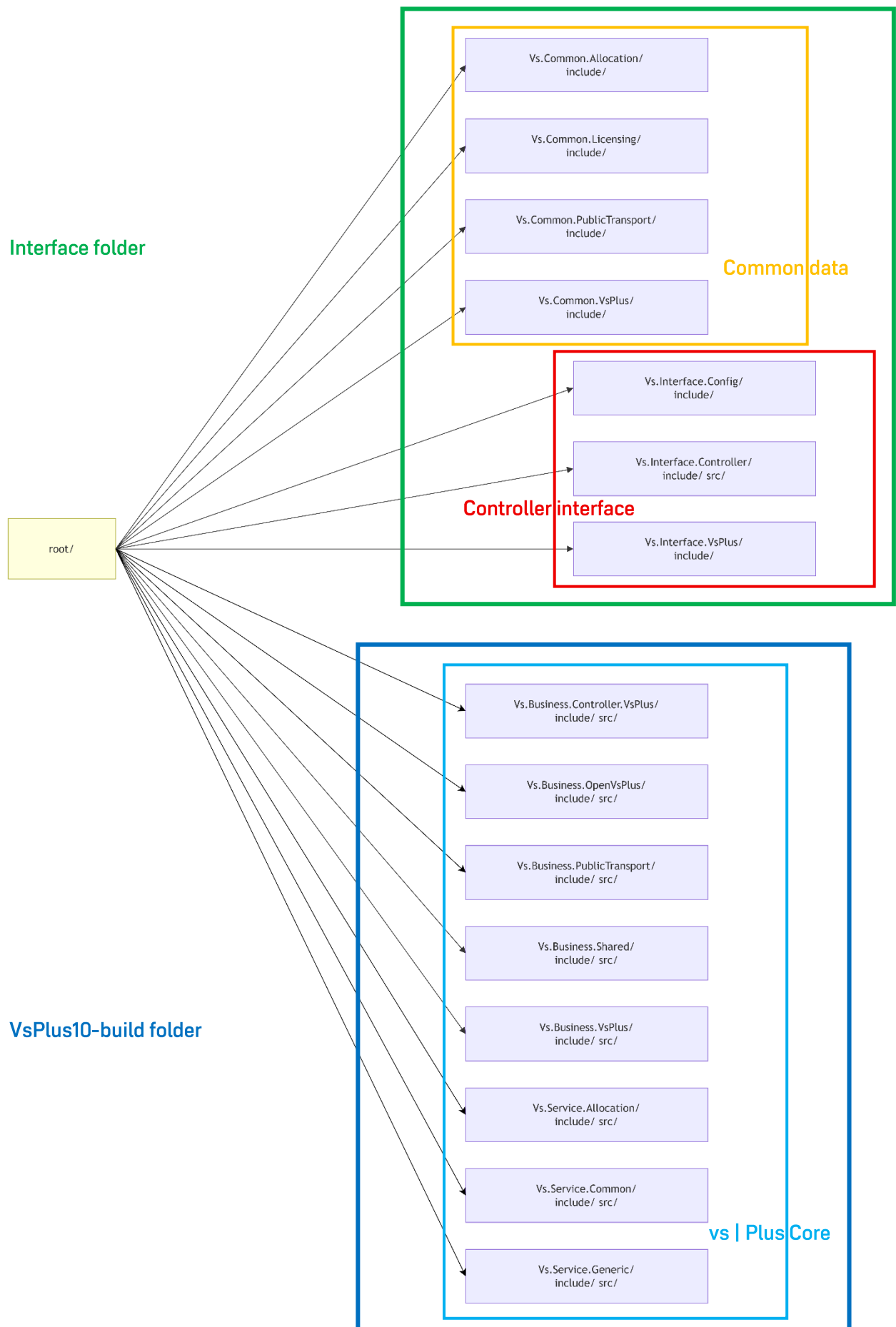
The interface functions have been expanded in version 10:

- New functions must be implemented according to instructions and inline comments
- The 4.5 listed functions must be checked and adjusted if necessary

Relevant files:

- **Integration of the controller → vs | plus:** VsplusControllerFunctions.c
- **Functions provided by vs | plus:** VsplusFunctions.h

Your adjustments take place in **the controller interface project directories**, while the vs | plus core remains unchanged.



3 BUILD

3.1 Compiler / Language Standard

Version 10 is created in C/C++ and requires at least the **C++11** language standard . The new **public transport module** is mainly implemented in C++ and is also based on this minimum standard.

Most other existing code areas are still implemented in C .

4 FUNCTIONAL INTERFACE EXTENSIONS

4.1 OCIT Color Codes

The current development of the vs | plus on-demand control supported OCIT color coding (hex values) can be found in the enumeration in OcitColorCodes.h (Vs.Common.VsPlus), according to the OCIT standard.

4.2 Newly Added Functions

4.2.1 Functions of the Signal Groups

4.2.1.1 Mapping a digital output to a signal group

Element	Description
Function Name	getSignalGroupChannelRelatedDigitalOutput
Signature	signed short getSignalGroupChannelRelatedDigitalOutput(signed short sg)
Short description	Determines the channel of the digital output associated with the specified signal group (according to OCIT mapping).
Parameters	sg – channel number of the signal group
Return Value	Channel number of the assigned digital output
Error case	Returns 0 if the specified signal group is invalid or no associated digital output exists

4.2.2 Digital Output Functions

4.2.2.1 Determination of the switch-on and switch-off time of a digital output

Element	Description
Function Names	getElapsedDigitalOn / getElapsedDigitalOff
Signatures	unsigned short getElapsedDigitalOn(signed short port) unsigned short getElapsedDigitalOff(signed short port)
Short description	Determine the elapsed time since the last time a digital output was turned on or off.
Parameters	port – channel number of the digital output
Return Value	Elapsed time since state change in Deci seconds
Behavior (On)	Returns the time since switched on; Return value 0 if the channel is invalid or the output is currently off
Behavior (Off)	Returns the time since switched off; Return value 0 if the channel is invalid or the output is currently on

4.2.2.2 Digital output access inspection

Element	Description
Function Name	isDigitalOutputEnabledForVsplus
Signature	unsigned short isDigitalOutputEnabledForVsplus(signed short port)
Short description	Checks if a digital output is available for access by vs plus. Analogous to access control for signal groups.
Parameters	port – channel number of the digital output
Return Value	Value > 0: Access Allowed; Value = 0: Channel Invalid or Access Not Allowed

4.2.2.3 Determination of the corresponding signal group of a digital output

Element	Description
Function Name	getDigitalOutputRelatedSignalGroupChannel
Signature	signed short getDigitalOutputRelatedSignalGroupChannel(signed short port)
Short description	Determines the signal group belonging to the specified digital output line according to OCIT mapping.
Parameters	port – channel number of the digital output
Return Value	Channel number of the corresponding signal group
Error case	Returns 0 if the channel is invalid or no associated signal group exists

4.2.3 Functions of On-demand Control**4.2.3.1 Checking supported OCIT color codes for a signal group**

Element	Description
Function Name	areOcitColourCodesSupported
Signature	unsigned short areOcitColourCodesSupported(signed short signalGroupChannel, OcitColourCodes* codes, int count)
Short description	Verifies that the passed OCIT color codes are configured and available for the specified signal group.
Parameters	signalGroupChannel – channel number of signal group codes – array of OCIT color codes count – number of elements in the array
Return Value	Value > 0: All color codes are available
Error case	Returns 0 if the signal group is invalid or at least one color code is not supported

4.2.3.2 Support for the deactivation of transitions in on-demand operation

Element	Description
Function Name	canDisableTransitionsWithinOnDemandControl
Signature	unsigned short canDisableTransitionsWithinOnDemandControl(void)
Short description	Checks whether the controller supports disabling state transitions in on-demand control.
Parameters	—
Return Value	Value > 0: Transitions can be disabled. In this case, the controller must be able to switch or suppress transitions for all signal groups.
Error case	Returns 0 if disabling transitions is not supported or only partially supported.

4.2.3.3 Checking the logical state of OCIT color codes of a signal group

Element	Description
Function Names	isOcitColorCodeSignalGroupLogicalStateOpen / isOcitColorCodeSignalGroupLogicalStateClosed
Signatures	signed short isOcitColorCodeSignalGroupLogicalStateOpen(signed short signalGroupChannel, OcitColourCodes ocitColourCode) signed short isOcitColorCodeSignalGroupLogicalStateClosed(signed short signalGroupChannel, OcitColourCodes ocitColourCode)
Short description	Check whether the logical state of a particular OCIT color code of a signal group corresponds to the “open” or “closed” state.
Parameters	signalGroupChannel – channel number of the signal group ocitColourCode – OCIT color code (enum value)
Return Value (Open)	Value > 0: logical state is “open” Value = 0: signal group or color code invalid or state is “closed”
Return Value (Closed)	Value > 0: logical state is “closed” Value = 0: signal group or color code invalid or state is “open”
Restriction	The “undefined” state is currently not supported. Return value 0.

4.2.3.4 Control of signal groups in on-demand operation (open / closed)

Element	Description
Function Names	switchSignalGroupOpenByOnDemandControl / switchSignalGroupClosedByOnDemandControl
Signatures	void switchSignalGroupOpenByOnDemandControl(signed short signalGroupChannel, int doIncludeTransition) void switchSignalGroupClosedByOnDemandControl(signed short signalGroupChannel, int doIncludeTransition)
Short description	Requests that a signal group be switched to the “open” or “closed” state during active on-demand control.
Parameters	signalGroupChannel – channel number of the signal group doIncludeTransition – Specifies whether or not a transition of the basic service (“open->closed” or “closed->open”) should be switched
Behavior (Open)	Switches the signal group to the “open” state. If the parameter is set, a transition from “closed” to “open” is executed.
Behavior (Closed)	Switches the signal group to the “closed” state. If the parameter is set, a transition from “open” to “closed” is executed.
Restriction	The functions are only called during active on-demand control

4.2.4 Functions of the New Public Transport Module

4.2.4.1 Reception of public transport request messages

Element	Description
Function Name	readPublicTransportRequestMessagePackage
Signature	signed short readPublicTransportRequestMessagePackage(PublicTransportMessagePackage* const message)
Short description	Reads an public transport request message packet from the controller's message queue.
Parameters	message – output structure, is filled with pointers to the message content as well as the message type (e.g. CAM, SREM, Serial R09)
Return Value	Value > 0: New message packet read successfully
Error case	Returns 0 if no new message is available or the package could not be set

4.2.4.2 Sending response messages (AMLi)

Element	Description
Function Name	sendPublicTransportMessageResponsePackages
Signature	void sendPublicTransportMessageResponsePackages(const PublicTransportMessageResponsePackage* const packages, int count)
Short description	Transmits an array of public transport response messages to the controller. The answers refer to previously read inquiries.
Parameters	packages – array of response packets (e.g., SerialR09 (AMLi), CAM, or SREM responses incl. type tagging) count – number of packets passed in
Return Value	—

4.3 Revised Existing Functions

4.3.1 Changed Signatures

4.3.1.1 Query of intergreen times of the controller

Element	Description
Function Name	getControllerIntergreen
Signature	signed short getControllerIntergreen(signed short sgclear, signed short sgenter)
Short description	Provides the intergreen times between two signal groups according to the current signal program in the basic supply of the controller.
Parameters	sgclear – channel number of the clearing signal group sgenter – channel number of the incoming signal group
Return Value	Intergreen time in Deci seconds
Error case	Invalid Signal Group(s) – Maximum Signed Short Value
	No intergreen time defined – maximum signed short value

4.3.2 Renamed Functions

4.3.2.1 Function change: getSignalGroupPartialNode → getSignalGroupPartialNodeId

Element	Description
Function Name	getSignalGroupPartialNodeId
Signature	unsigned short getSignalGroupPartialNodeId(signed short signalGroupChannel)
Short description	Returns the ID of the partial node (traffic engineering defined) to which the signal group is assigned.
Parameters	signalGroupChannel – channel number of the signal group
Return Value	Valid partial node ID [1..max]; in case of invalid or unassigned signal group: 0

4.3.2.2 Function change: isSignalGroupInOffStateColour → isOnDemandColourActiveForSignalGroup

Element	Description
Function Name	isOnDemandColourActiveForSignalGroup
Signature	unsigned short isOnDemandColourActiveForSignalGroup(signed short signalGroupChannel, OcitColourCodes ocitColourCode)
Short description	Checks whether a signal group currently has the specified color active.
Parameters	signalGroupChannel – channel number of the signal group
	ocitColourCode – OCIT colour code
Return Value	> 0 if active; 0 for invalid parameters or other current color

4.3.2.3 Function change: switchSignalGroupOffStateColour → switchSignalGroupOnDemandColour

Element	Description
Function Name	switchSignalGroupOnDemandColour
Signature	void switchSignalGroupOnDemandColour(signed short signalGroupChannel, OcitColourCodes ocitColourCode, int doIncludeTransition)
Short description	Prompts the controller to switch a signal group to a specific color.
Parameters	signalGroupChannel – channel number of the signal group
	ocitColourCode – OCIT colour code
	doIncludeTransition – Consider transition or not
Return Value	—

4.3.2.4 Function change: getElapsedOffStateColour → getElapsedOnDemandColour

Element	Description
Function Name	getElapsedOnDemandColour
Signature	unsigned short getElapsedOnDemandColour(signed short signalGroupChannel, OcitColourCodes ocitColourCode)
Short description	Returns the elapsed time since switching the signal group to the specified color.
Parameters	signalGroupChannel – channel number of the signal group
	ocitColourCode – OCIT colour code
Return Value	Time in Deci seconds
Error case	Invalid parameters or other color active – return value 0

4.4 Removed Functions

4.4.1 Unused Functions

The following functions have not been supported by the controller interface for a long time, or have not been supported by vs | plus and have therefore been removed:

- signed short getDetSSCounter(const signed short det)
- void clearDetSSCounter(const signed short det)

4.4.2 Relays, Flashers and Digital Outputs

By merging the “relay” and “flasher” ports into digital outputs, the previous flasher functions are no longer required and have been removed from the interface:

- void switchOnBlinker(const signed short channel)
- void switchOffBlinker(const signed short channel)
- signed short isOnBlinker(const signed short channel)
- signed short isOffBlinker(const signed short channel)

4.4.3 Replacement of the Old Public Transport Module

The new public transport module enables the reception of various public transport messages or telegrams. The function that had offered pure serial R09 transmission has therefore been removed:

- signed short readPTMessage(void *const msg)

4.5 Potentially Necessary Corrections on Controller Site

The subsequent implementations of the functions for providing the signal group data must be reviewed and corrected if necessary, if they do not meet the requirements defined below.

Standard state corresponds to the “Open” state defined in the basic supply , or “Closed” of a signal group.

4.5.1 Functions isOpen / isClosed

The “Open” and “Closed” states refer exclusively to the respective **standard states and colors** of the signal groups.

Deviating operating states, especially those from demand control (e.g. “flashing yellow”), must be taken into account accordingly: A color that functionally signals the “Open” state , but **does not correspond to the defined standard color for “Open”**, may not be interpreted as **“true” by the “isOpen” function**.

The **function “isOpen”** may only return the value “true” if the active color actually **corresponds to the specified standard state “Open”**.

4.5.2 Functions isOpenMin / isClosedMin

The **“isOpenMin” and “isClosedMin” functions** must indicate whether the signal group is in the respective standard state for at least the defined **minimum duration** .

Similar to “isOpen” and “isClosed”, these functions refer exclusively to the defined **standard states and colors**.

If a color (e.g. “flashing yellow”) is defined as **the standard state**, the period must be fully taken into account when checking the minimum duration.

If the signal group is in a **deviating state or color** that does not meet the defined standard, this period may **not** be included in the minimum duration; in this case, the timing must **be reset**.

The functions may only return the value “true” if the signal group meets the color of the standard open or standard closed state **without interruption** within the defined minimum time.

4.5.3 Functions getElapsedOpen / getElapsedClosed

The functions “getElapsedOpen” and “getElapsedClosed” must return the duration during which the signal group is in the respective **standard state**.

Analogous to the states “isOpen” and “isClosed”, these functions refer exclusively to the defined **standard states and colors**.

If a state such as “flashing yellow” is defined as the **standard state**, the time duration must be updated and returned accordingly.

If, on the other hand, the signal group is in a **deviating state or color** that does not correspond to the defined standard, the time may **not be updated any further**; in this case, the time measurement must be **reset**.

The returned duration is therefore only valid for periods in which the active signal phase actually corresponds to the specified **standard state**.

4.5.4 Functions getElapsedOpenMin / getElapsedClosedMin

The functions “getElapsedOpenMin” and “getElapsedClosedMin” must return the currently accumulated period of time during which the signal group is in the respective **standard state with respect to the minimum time**.

Time measurement refers exclusively to **contiguous time intervals** in the defined standard state.

If a state (e.g. “flashing yellow”) is defined as the **standard state**, the time for this state is updated accordingly.

If, on the other hand, the signal group is in a **deviating state or color** that does not correspond to the defined standard, the time may **not be updated any further**; in this case, the time measurement must be **reset**.

The returned duration serves as a basis for evaluating how far **the respective minimum time has progressed**.

5 MEMORY ALLOCATION EXTENSIONS

5.1 Expansion of Memory IDs

New memory area IDs have been added to the **MemoryIdentifiers.h** file (Vs.Common.Allocation).

Since version 9, vs | plus has an internal memory manager that can manage up to **500 memory areas** in the controller. The possible IDs range over the entire integer value range (min to max int).

5.2 Runtime Restrictions

The new public transport module is developed in C++ and uses the STL (**Standard Template Library**). There is a central restriction here: After initializing vs | plus, **no more memory may be dynamically allocated or released**.

5.3 Memory Access Concept and STL Usage

In order to enable the use of STL containers, a **larger memory block** is reserved at the start .

- Access is via special **allocators**
- Use of pre-reserved memory using **placement new/delete**
- No dynamic memory operations required at runtime

This concept enables efficient use of the STL under the given system restrictions. The public transport module is the first module in vs | plus, which uses this principle. Currently, **1 MB (1,048,576 bytes)** is reserved for this purpose .

5.4 New Memory IDs for the Public Transport Module

The following IDs have been defined for the public transport module:

- MEMORY_POOL = 0x7FFFFFFF
- MEMORY_POOL_FREE_LISTS = 0x7FFFFFFE
- MEMORY_POOL_INTERNAL = 0x7FFFFFD
- PUBLIC_TRANSPORT_DATA_PROVIDER = 0x7FFFFFC
- PUBLIC_TRANSPORT_PROCESSOR = 0x7FFFFFB
- PUBLIC_TRANSPORT_RUNNER = 0x7FFFFFA

6 THE NEW PUBLIC TRANSPORT MODULE

Version 10 completely replaces the previous public transport module with a new implementation. The old module was based on a third-party solution that had been used reliably over a longer period of time, but no longer met the current technological and conceptual requirements.

The new solution was developed entirely in C++ (at least C++11) and uses its modern language features to improve maintainability, extensibility and code quality.

The previous functionality was adopted, conceptually revised and modernized in a targeted manner. The old module is no longer supported.

6.1 Data Structures and Formats

All necessary data structures and definitions, including for SREM and CAM messages, are contained in the subproject **Vs.Common.PublicTransport** . They comply with the applicable OCIT specification.

6.2 Public Transport Requests

During one processing cycle, vs | plus calls the function `readPublicTransportRequestMessagePackage` several times (currently up to five times).

If there are public transport requests (serial R09, CAM or SREM), these are provided in advance in a structured form as **packages** .

One package contains:

- exactly one valid request
- the corresponding type (R09, CAM, or SREM) of the request

6.3 Public Transport Responses

Public transport responses are sent by vs | plus in a “push” manner once a processing cycle using the function `sendPublicTransportMessageResponsePackages`.

The controller decides for itself whether to read out the incoming responses. To increase efficiency, only structured values (no more strings as provided by the old module) are transmitted.

Again, the following applies:

- A package contains exactly one response
- and includes the associated type (R09 (AMLi), CAM or SREM)

6.4 AMLi Records (Extended Serial R09)

The previous concept of a public transport memory module storing up to 999 AMLi records in “text” format strings has been discarded.

Instead, the new module will:

- Generate a maximum of one AMLi record per public transport request
- output the record directly via the device interface (see. Public Transport)

This completely eliminates the need for intermediate storage and later readout via direct access.

7 EXPANSION OF PROCESS DATA

7.1 Extended New Public Transport Module Data

With the introduction of the new public transport module, the available process data has been significantly expanded.

A complete C-style list of all process data IDs as well as the maximum number of related topology objects is defined in `ProcessDataList.h` (Vs.Common.VsPlus).

The detailed descriptions of the individual process data can be found in the same subproject in the file `ProcessDataDefinitions.h`.

The newly available process data is also retrieved via the already existing and familiar interface function. The instance ID is expected to be the logical detector / reporting point or traffic stream ID.

7.2 Replacement of Historical Line-Course IDs with Process Data in the PT module

Name	Description
OITD_OEV_MODIFIKATION_VS	Replaces the former line-course ID 88888 for a modification
OITD_PTMODULE_EMERGENCY_CHECKIN_ACTIVE	Replaces the former line-course ID 77777 for emergency requests
OITD_OEV_NOTAN_WARTEZEIT_VS	Complements the replacement of the 77777 ID (emergency request), e.g. to display the waiting time
OITD_OEV_ZWANGSABM_VS	Replaces the former line-course ID 99999 for forced check-out of a public transport

7.3 General Process Data Output for Public Transport Inquiries

Name	Description
OITD_PUBLIC_TRANSPORT_REQUEST_CHANNEL	OCIT-Outstation (STG) channel of the reporting point
OITD_PUBLIC_TRANSPORT_REQUEST_TYPE	Type of submitted public transport request (PublicTransportRequestTypes)
OITD_PUBLIC_TRANSPORT_NOF_REQUESTS	Number of transmitted public transport requests in the current vs plus cycle
OITD_PUBLIC_TRANSPORT_REQUEST_YEAR	Year of the PT request timestamp [0..99]
OITD_PUBLIC_TRANSPORT_REQUEST_MONTH	Month of public transport request timestamp [1..12]
OITD_PUBLIC_TRANSPORT_REQUEST_DAY	Day of the public transport request timestamp [1..31]
OITD_PUBLIC_TRANSPORT_REQUEST_HOUR	Hour of Public Transport Request Timestamp [0..23]
OITD_PUBLIC_TRANSPORT_REQUEST_MINUTE	Minute of the public transport request timestamp [0..59]
OITD_PUBLIC_TRANSPORT_REQUEST_SECOND	Second of the PT request timestamp [0..59]

7.4 Public Transport Process Data (Extended R09 / AMLi)

Name	Description
OITD_R09EXT_TIMESTAMP_YEAR	Year of R09 timestamp [0..99]
OITD_R09EXT_TIMESTAMP_MONTH	Month of R09 timestamp [1..12]
OITD_R09EXT_TIMESTAMP_DAY	Date of R09 Timestamp [1..31]
OITD_R09EXT_TIMESTAMP_HOUR	Hour of R09 Timestamp [0..23]
OITD_R09EXT_TIMESTAMP_MINUTE	Minute of R09 timestamp [0..59]
OITD_R09EXT_TIMESTAMP_SECOND	Second of R09 timestamp [0..59]
OITD_R09EXT_CALLPOINT_NR	Reporting point number of the R09 receipt [1..16777216]
OITD_R09EXT_TIMETABLE_DEVIATION	Timetable deviation in seconds [-3599..3599]
OITD_R09EXT_LINE	Vehicle line number [0..999]
OITD_R09EXT_ROUTE	Vehicle route number [0..999]
OITD_R09EXT_TOUR	Course number of the vehicle [0..99]
OITD_R09EXT_PRIORITY	Priority of the vehicle [0..7]
OITD_R09EXT_TRAIN_LENGTH	Train length of the vehicle [0..7]
OITD_R09EXT_MANUAL_DIRECTION	Manual direction indication of the vehicle [0..3]
OITD_R09EXT_RELATIVE_INTERSECTION	Relative node identifier [0..255]
OITD_R09EXT_IS_MODIFICATION_BY_CENTER	Specifies whether a modification is made by the control panel [0 (no), 1 (yes), 255 (undefined)]
OITD_R09EXT_RECEPTION_CYCLE_SECOND	Cycle second of R09 reception [1..255]
OITD_R09EXT_SIGNAL_PLAN	Currently running signal plan / program [0..32]
OITD_R09EXT_CURRENT_PHASE	Current phase of the associated signal group [0 (none)..255]
OITD_R09EXT_DESIRED_PHASE	Desired phase of the associated signal group [0 (no transition active/no phases)..255]
OITD_R09EXT_TRAVEL_TIME	Vehicle travel time [0 (not available / emergency request) .. 255]
OITD_R09EXT_SIGNAL_GROUP_OPEN_STARTED	Red end of the assigned signal group [0..255]
OITD_R09EXT_SIGNAL_GROUP_OPEN_STOPPED	Green end of the assigned signal group [0..255]

7.5 Digital Public Transport Process Data (Digital R09 / SREM/SSEM)

Name	Description
OITD_R09DI_REPORTING_POINT	Reporting point number of the digital R09 reception [0..65535]
OITD_R09DI_PRIORITY_LEVEL	Priority of the vehicle [0..255]
OITD_R09DI_LENGTH	Train length of the vehicle [0..7]
OITD_R09DI_ROUTE	Vehicle route number [0..65535]
OITD_R09DI_LINE	Vehicle line number [0..65535]
OITD_R09DI_DIRECTION	Manual Vehicle Direction [0..255]
OITD_R09DI_TOUR	Course number of the vehicle [0..4294967295]
OITD_R09DI_VERSION	Basic version of public transport supply [0..4294967295]

7.6 Additional Public Transport Process Data (CAM – Cooperative Awareness Message)

Name	Description
OITD_CAM_STATION_ID	Requestor ID/Vehicle
OITD_CAM_STATION_TYPE	Type of requestor / vehicle
OITD_CAM_POSITION_LATITUDE	Position (latitude) of the requestor / vehicle
OITD_CAM_POSITION_LONGITUDE	Position (longitude) of the requestor / vehicle
OITD_CAM_EMBARCATION_STATUS	Requestor / Vehicle has opened doors (Yes / No)

7.7 Additional Public Transport Process Data (SREM – Signal Request Extended Message)

Name	Description
OITD_SREM_STATION_ID	Requestor ID/Vehicle
OITD_SREM_BASIC_VEHICLE_ROLE	Type of requestor / vehicle
OITD_SREM_TRANSIT_SCHEDULE	Timetable deviation in seconds (*10) [-122..121] (-122 = not available)
OITD_SREM_TIMESTAMP	Current year minute (UTC) [0..527040] (527040 = invalid)
OITD_SREM_TIMESTAMP_SECOND	Millisecond within the current UTC minute [0..65535]
OITD_SREM_HEADING	Direction of requestor / vehicle
OITD_SREM_POSITION_LATITUDE	Position (latitude) of the requestor / vehicle
OITD_SREM_POSITION_LONGITUDE	Position (longitude) of the requestor / vehicle
OITD_SREM_SPEED	Speed of requestor / vehicle

7.8 General Public Transport Module Process Data (Type-independent)

Name	Description
OITD_PTMODULE_CALLPOINT	Reporting point number regardless of the type of public transport request
OITD_PTMODULE_VEHICLE_ID	Vehicle-ID (requestor) regardless of the type of public transport request
OITD_PTMODULE_VEHICLE_TYPE	Vehicle type (requestor) regardless of the type of public transport request
OITD_PTMODULE_VEHICLE_LENGTH	Train length regardless of the type of public transport request
OITD_PTMODULE_PRIORITY	Priority regardless of the type of public transport request
OITD_PTMODULE_TIMETABLE_STATUS	Timetable situation regardless of the type of public transport request
OITD_PTMODULE_LINE	Line regardless of the type of public transport request
OITD_PTMODULE_TOUR	Course independent of the type of public transport request
OITD_PTMODULE_ROUTE	Route regardless of the type of public transport request
OITD_PTMODULE_DIRECTION	Manual direction regardless of the type of public transport request
OITD_PTMODULE_EMERGENCY_CHECKIN_ACTIVE	Acknowledgment of the emergency request (for synoptics)
OITD_PTMODULE_POSITION_LATITUDE	Position (latitude) independent of the type of public transport request
OITD_PTMODULE_POSITION_LONGITUDE	Position (longitude) regardless of the type of PT request
OITD_PTMODULE_SPEED	Speed regardless of the type of public transport request

8 ONGOING ENHANCEMENTS AND CORRECTIONS

8.1 Improvements from vs | plus 9.1.0 (Patch 6)

Applies to improvements made in version 9.1.0 after the release of the initial version 10.0.0 and subsequently ported.

U 10474	Enhancements and Fixes for vs plus On-demand Control (Patch 6)
	Starting with version 9.1.0 [2][1] (Patch 6), the following issues with on-demand control were resolved: - Correction of the minimum time monitoring when a controller is not able to distinguish between standard and non-standard open/closed times. - Fixed a value overflow that could cause a significant delay in returning to on-demand control. - Removed a duplicate call to on-demand control when supplying new parameters during runtime. <i>In addition, the following enhancement was implemented:</i> On-demand control now supports staggered starts, thereby enabling use cases such as the pre-start of signal groups. To this end, the minimum time monitoring was extended from sub-sequences to the sequence level.
U 8941 U 10107	Adjust the supply of on-demand control sequences
	<i>Regarding parameterization and VCB exports with the vs worksuite:</i> Starting with version 9.1.0 [2][1] (Patch 6), when supplying vs plus parameters, sequences stored in the VCB for on-demand control are ignored if on-demand control is not enabled in any program definition. Previously, this resulted in a supply error.
B 10368	Exception handling for “open vs plus” script errors
	Processing an open vs plus script containing errors that were not detected by the interpreter until runtime resulted in an exception after the interpreter terminated and a warning message No. 50 was displayed. Starting with version 9.1.0 [2][1] (Patch 6), a subsequent access of the interpreter is suppressed.

8.2 Improvements in Version 10.0.0

Applies to improvements introduced since the release of the initial version 10.0.0.

B 10599	Parallel Public Transport Requests Delete Existing Registrations For public transport callpoints of type "parallel", it was not verified whether a request from the assigned detector was actually present. As a consequence, the callpoint-sequence was continuously checked-out, causing existing check-in requests to be removed. This issue has been corrected as of version 10.0.0[0][1].
B 10601 / B 10654	Request Triggered by open vs plus Public Transport Telegram Not Working In certain cases, public transport telegrams originating from open vs plus were not processed correctly because they were removed from memory before the callpoint-sequences were evaluated. As of version 10.0.0[0][1], the telegrams are correctly buffered and merged with the remaining requests when updating the other public transport inputs. As a result, processing of public transport telegrams from open vs plus is ensured again.
11/09/2026	Adjust Output of Internal vs plus Version Information As of version 10.0.0[0][1], the internal version numbering of vs plus has been changed back from the format N.N.N (Major, Minor, Patch) to the previously used format (N.N.N[N][N]). In addition, new sub-version identifiers have been defined in the common interface: - VSP_INTERFACE_VERSION - VSP_MAJOR_VERSION - VSP_MINOR_VERSION - VSP_FEATURE_PATCH_VERSION - VSP_PATCH_VERSION The output format is as follows: Interface.Major.Minor[FeaturePatch][Patch]
U 10530	Align Line and Route Supply with vs studio Starting with vs studio 3.2, specifying "<i>Lines and Routes</i>" for public transport supply is mandatory. This makes it clear to the user that the public transport module operates with line and route tracking. Data completeness is already validated during parameterization. As of vs plus 10.0.0[0][1], error message no. 13 is no longer generated when supply-data is received without line and route information. In such cases, the missing information is interpreted as "all lines – all routes".
U 9832	Consider Rest-Time and Check-out Delay-Time for Serial Telegrams The behavior of rest- and delay-times within the public transport module has been reviewed. The rest-time for serial public transport telegrams is not used. The delay-time for serial check-in telegrams remains unchanged and corresponds to the behavior of version 9.1.0. An adjustment of the delay-time for check-out telegrams will not be implemented for the time being, as the expected benefit does not justify the required implementation effort. Standardization will be carried out at a later stage as part of the integration of Standard Public Transport into the Public Transport Module.

B 10633	Relax Validation of Supplied Public Transport Memory Callpoint-Sequences
	In certain configurations, supply error message no. 13 was incorrectly generated when the number of traffic stream based callpoint-sequences differed from the number of callpoint based callpoint-sequences.
	The validation failed to consider that traffic stream based callpoint-sequences without an assigned callpoint based callpoint-sequence are valid and can be processed by vs plus independently of the public transport module.
	As of version 10.0.0[0][1] , the supply validation has been adjusted accordingly, ensuring that only genuinely invalid configurations are reported as errors.

Imprint

File name	vsplus Version 10 Release Notes EN.docx
Save date	02.10.2026 09:58
Last Modified	Cyrill Jucker
Quality assurance	SQS-certified quality system according to ISO 9001 (Reg.Nr. 34856)
© Copyright	vs Verkehrssysteme AG