Volvo Car Corporation

Document Release Status
RELEASED

Release Date
2013-12-11

Version

Document No
31843799

Revision
014

Volume No
01

Page No
1 (104)



Document Type
**NOTE-SWRS**

Reference No

Document Name
**VCC IP COMMAND PROTOCOL**
**v1.9**
**SPA**

Part must comply with VCC Restricted Substance Management Standard (RSMS) VCS 5036,5

NB! Change the meta data in the header by pressing "Press here to update this document's meta data" in the menu "TechNotes_VCC". The header is automatically updated. (Do not remove this text!)

# Software Requirement Specification

## VCC IP COMMAND PROTOCOL
## AND
## VCC IP LINK MANAGER
## SPA
## V1.9
### Version 1.9

Issued/Created in Elektra by (Dept, Name, Phone, Id)
94141, Magnus Eek, +46-31-3255925, MEEK1

Changes since last revision
See Document revision history.

# Contents

# 1. Introduction

## 1.1 What is the purpose of this document?

This document defines the VCC IP (Internet Protocol) Command Protocol which is defining a control channel for IP based networks (Ethernet STP, APIX2, USB or MOST150). This control channel is called the "VCC IP Command Protocol". The protocol is designed for in-vehicle communication, between IP-based ECUs. This document does not define the application signals (services), which is used by the control messages between the clients and servers. Instead, this document defines the rules on how to the control communication shall be handled between the applications.
In addition, this document also includes a chapter defining an IP link management mechanism called “VCC IP Link Manager”, LM, with the purpose of providing up-to-date status information about IP interfaces within the vehicle and a way of enabling the vehicle internal IP ECUs, thus waking up the IP links. LM is based on the VCC IP Command Protocol.

This document shall be used together with the corresponding signal document (for a specific VCC project) which defines the signal content (the actual application data).

## 1.2 Who should read this document?

This document is mainly intended for designers/developers of the signal content on the application layer. The designer may be an external supplier or a development department (e.g. VCC function owners) within Volvo Car Corporation. Additionally, test engineers should read this document to have an understanding of the mechanism and behavior of the control messages which is used between IP-based ECUs within the vehicle.

Requirements within this specification are defined as: [VCC IP Prot: XXXX/;-revision], where XXXX is the requirement number.

## 1.3 How is this template document organized?

Chapter one contains general information and chapter two defines the VCC IP Command Protocol and chapter three defines the VCC IP Link Manager.
Chapter four contains the appendix.

## 1.4 How do you receive more information?

Contact the document owner at the department Vehicle Network Design, 94141 at Volvo Car Corporation.

## 1.5 Document revision history

| Revision | Date | Description | Author (dept, name, phone) |
|---|---|---|---|
| 001-01 | 2012-04-20 | Template for IP based API | 94141, Magnus Eek (meek1) |
| 001-02 | 2012-05-31 | New Template for IP based API | 94141, Magnus Eek (meek1) |
| 001-03 | 2012-06-08 | Redesign and update of chapter 1. | 94141, Magnus Eek (meek1) |
| | 2012-06-09 | Renamed RequestNoReturn to SetRequestNoReturn | Actia, Andreas Bergvall |
| 001-03 | 2012-06-12 | * Updated OperationType table<br>* Updated definition table<br>* Added new overview list of OperationIDs for ServiceID<br>* New datatypes<br>* New improved examples in Appendix A | 94141, Magnus Eek (meek1) |
| 001-04 | 2012-06-28 | * General, restruction of some subchapters. E.g. moved timers, message handling to separate chapters.<br>* 1.8, Added new abbreviations<br>* 1.10.1, updated body text<br>* 1.10.1.2, updated body text.<br>* 1.10.2.2.5, updated body text<br>* 1.10.2.2.7, updated descriptions<br>* 1.10.2.2.7.1.1, updated body text<br>* 1.10.2.2.7.1.2, updated * SetRequestNoReturn. Removed optional Error transmission. Added rules<br>* 1.10.2.2.7.2 updated body text and figure in Notifications.<br>* 1.10.2.2.7.3, updated body text<br>* 1.10.2.2.8, updated body text<br>* 1.10.2.2.9, updated typo in the example<br>* 1.10.2.2.10, updated body text regarding the encoding of the payload.<br>* 1.10.2.3, updated body text and subchapters containing different timers and figure<br>* 1.10.2.3.1, updated new body text regarding Acknowledgement timer<br>* 1.10.2.3.1.1, updated example for "wait-for-ack"<br>* 1.10.2.3.2, updated new body text regarding Response timer<br>* 1.10.2.3.2.1, updated example for "wait-for-response"<br>* 1.10.2.4, new chapter regarding messagehandling<br>* 1.10.2.4.1, new chapter regarding retransmission of messages | 94141, Magnus Eek (meek1) |
| 001-05 | 2012-09-13 | * 2.1.2.4.2, updated Figure 11<br>* 2.1.2.4.3, updated Figure 14<br>* 2.1.2.4.4, updated Figure 17<br>* 2.1.2.4.7, updated Figure 24<br>* **Error! Reference source not found.**, updated Figure 9<br>* **Error! Reference source not found.** - updated body text. | 94141, Magnus Eek (meek1) |

| | | | |
|---|---|---|---|
| | | * 2.1.3.1.3 - updated Acknowledgement Timer body text with defaultTimeoutWFA value.<br>* **Error! Reference source not found.** - updated body text and Figure 11.<br>* 2.1.3.1.4 - updated bodytext and "WFR" Timer with defaultTimeoutWFR value.<br>* **Error! Reference source not found.** - updated bodytext and Figure 12.<br>* 2.1.3.1.4.4 - updated bodytext<br>* 2.1.3.3 - updated bodytext<br>* 2.1.3.3.2.1 - updated bodytext, clarification of the retransmission formula.<br>* 2.1.3.3.3 – updated **Error! Reference source not found.**.<br>* 2.1.4 – updated bodytext and clarification of error code rules. Updated table 6 with new error code, 0x06 – Busy and changed the reserved number interval for generic errors.<br>* 2.1.4.1.2, new section containing god practice of handling error codes. | |
| 001-06 | 2012-09-26 | * updated Message Handling, chapter 2.1.3.1.4.4.<br>-Clarification that both SenderHandle + IP address shall be used to distinguish between different messages.<br>- Added a new note<br><br>* Updated note within static notifications, chapter **Error! Reference source not found.**.<br>* Updated figure Message Sequence Format - Static Notification.<br>* Updated defaultTimeoutWFA value, chapter 2.1.3.1.3.<br>* Updated body text in Example- Wait-for-Ack, chapter **Error! Reference source not found.**<br>* Updated defaultTimeoutWFR value, chapter 2.1.3.1.4.<br>* Updated retransmission values (numberOfRetriesWFx), chapter 2.1.3.3<br>* Updated increaseTimerValueWFx values, chapter 2.1.3.3.2.1<br>* Created example showing WFA and WFR retransmission values, chapter **Error! Reference source not found.**<br>* Extended General Error Codes-* ErrorInformation. Increased from 8bit to 16bit, chapter 2.1.4<br>* Updated example Incorrect header, 2.1.4.1.2.2 | 94141, Magnus Eek (meek1), Volvo Car Corporation |
| 01-007 | 2012-10-12 | * The VCC IP Protocol is moved to a new separate document, containing no application signal content. | 94141, Magnus Eek (meek1), Volvo Car Corporation |

| | | | |
|---|---|---|---|
| | | * Added external reference to ASN.1 syntax<br>* 2.1 - updated body text<br>2.1.1.4 - added new note<br>* 2.1.2 - updated body text<br>* 2.1.2.3.5 - updated body text<br>* 2.1.2.4.2 - updated body text<br>* 2.1.2.4.4 - updated body text<br>* **Error! Reference source not found.** - updated body text and created new subchapters<br>* 2.1.3.1.3 - updated body text<br>* 0 - new subchapter containing default WFA-values<br>* 2.1.3.1.4 - updated body text<br>* **Error! Reference source not found.** - new subchapter containing default WFR-values<br>* 2.1.3.1.4.4 - increased concurrent sequence value to minimum 10.<br>* 4 – new chapter containing information on ASN.1 syntax. | |
| 01-001 | 2012-11-28 | The document is handled and released as a new separate VCC document, using document nr: 31843799.<br>Therefore, the volume and revision counting is rested.<br><br>Matches the VCC IP8 release.<br><br>Major restructure of the document. Document now contain clear requirements. | 94141, Magnus Eek (meek1), Volvo Car Corporation |
| 01-002 | | Minor template adjustments.<br><br>Matches the VCC IP8 release. | 94141, Magnus Eek (meek1), Volvo Car Corporation |
| 01-003 | 2013-04-08 | VCC IP Prot: 0011/;1<br>- spellcheck performed<br><br>VCC IP Prot: 0019/;1<br>- New protocol number: 0x01<br><br>VCC IP Prot: 0020/;1<br>- Spellcheck performed on descriptions in table.<br><br>2.1.2.4 Application Message Format Sequences<br>- Updated erroneous cross-reference<br><br>VCC IP Prot: 0024/;1<br>- Removed the 'ACK' message upon reception of a Error Message.<br><br>VCC IP Prot: 0026/;1<br>- Updated erroneous cross-reference<br><br>VCC IP Prot: 0027/;1<br>- Updated figure 12 and removed the client | 94141, Magnus Eek (meek1), Volvo Car Corporation |

| | | | |
|---|---|---|---|
| | | responsibility to send a ACK message upon reception of a Error message.<br><br>VCC IP Prot: 0028/;1<br>- Updated erroneous cross-reference<br><br>VCC IP Prot: 0030/;1<br>- Updated figure 15 and removed the client responsibility to send a ACK message upon reception of a Error message.<br><br>VCC IP Prot: 0032/;1<br>- Updated figure 18 and removed the client responsibility to send a ACK message upon reception of a Error message.<br><br>VCC IP Prot: 0035/;1<br>- Clarification of requirement on NOTIFICATION messages.<br><br>VCC IP Prot: 0056/;1<br>- Update and clarification of requirement.<br><br>VCC IP Prot: 0057/;1<br>- Update and clarification of requirement.<br><br>VCC IP Prot: 0058/;1<br>- Update of value "numberOfRetriesWFA"<br><br>VCC IP Prot: 0063/;0<br>- New requirement defining/clarifying the defaultTimeoutWFx<br><br>Chapter 2.1.3.3.5.1 WFx-sequence<br>- Updated erroneous cross-reference<br><br>Chapter 2.1.4 General Error Codes<br>- Updated bodytext and removed sentece "The server shall always send an ACK, before..." | |
| 01-005 | 2013-06-26 | Change defaultTimeoutWFR from 300 ms to 500 ms<br><br>[VCC IP Prot:0063/;-0] – Removed reqirement since it caused inconsistency.<br><br>[VCC IP Prot: 0062/;-1] – Clarify interpretation of ErrorCode BUSY.<br><br>[VCC IP Prot: 0060/;-1] – Clarified the possibility to uniquely specify defaultTimeoutWFR and numberOfRetriesWFR for each VCC IP Command signal. | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| 01-006 DRAFT | 2013-07-05 | Adding VCC IP Link Manager and updated rest of the document to adapt to this | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |

| | | | |
|---|---|---|---|
| | | addition. | |
| 01-006 REL | 2013-08-01 | eT: 10208016<br>3.1.1.4.8 [VCC IP Prot: 0086/;-0] : Rewrite the Purpose description.<br><br>3.1.1.5.2 [VCC IP Prot: 0088/;-0] : Added requirement text to cover the case when an LSC releases the IP bus while being used by other LSCs. | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| 01-006 REL | 2013-08-14 | Updated requirements descriptions.<br><br>3.1.1.1.5 [VCC IP Prot: 0074/;-0] : Added ACTION type AVAIL_REQ.<br><br>3.1.1.4.5 [VCC IP Prot: 0083/;-0] : Added second ARS bullet.<br><br>3.1.1.6.3 [VCC IP Prot: 0091/;-0] : Updated naming of priority parameter. | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| 01-006 REL | 2013-09-23 | Updated requirement.<br>3.1.1.4.1 [VCC IP Prot: 0080/;-1] : Changed so timer *Request_monitoring_timeout* should not be specified in CarConfig.<br><br>New requirement.<br>3.1.1.4.2 [VCC IP Prot: 0099/;-0] : New, specifying the need for variable timer *Request_monitoring_timeout* and how it is decided. | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| 01-006 REL | 2013-09-23 | Increased revision number of document according to VCC document release tool. | 94141, Magnus Eek (meek1) Volvo Car Corporation |
| 01-007 REL | 2013-09-26 | 2.1.2.3-> changed the "Reserved I" field to "DataType"<br><br>2.1.2.3.1 -> changed the "Reserved I" field to "DataType"<br><br>2.1.2.3.8 -> Updated text and clarified the usage of "DataType" with a new requirement [VCC IP Prot: 0094/;-0] .<br><br>2.1.2.3.10.2, Updated [VCC IP Prot: 0022/;-1], clarified if the control message shall be encoded, the ASN.1 PER-Unaligned shall be used.<br><br>Updated 2.1.2.4.2.3 Design guidelines, replaced "ReservedI" with "DataType"<br><br>New protocol version according to requirement [VCC IP Prot: 0019/;-1] | 94141, Magnus Eek (meek1) Volvo Car Corporation |
| 01-007 REL | 2013-09-31 | Updating requirements surrounding notifications.<br>The two types of notifications described as cyclic and event-based has been separated | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |

| | | | |
|---|---|---|---|
| | | into two different operation types to easier distinguish them from each other.<br>Added requirements stating the usage of cyclic notifications with regards to ACKs.<br><br>New requirements;<br>2.1.2.4.6.3.2 [VCC IP Prot: 0095;-0]<br>2.1.2.4.6.4.2 [VCC IP Prot: 0096;-0]<br><br>Updated requirements;<br>2.1.2.4.3.1.1 [VCC IP Prot: 0030/;-1]<br>3.1.1.1.4 [VCC IP Prot: 0073/;-0]<br>All within paragraph;<br>2.1.2.4.5 Notification message Requirements<br>2.1.2.4.6 OperationType Notification and Cyclic Notification<br>2.1.2.4.7 OperationType NotificationRequest | |
| 01-007 REL | 2013-10-01 | Added new chapter:<br>4.2.1 Range/Sign to Data Type Specifying a table mapping ranges and signs (specified in the signals specification) to specific data types. | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| v1.8 DRAFT | 2013-10-09 | eT: 10212130<br>Updated according to requested eTracker, adding the function to request specific Resource Groups using the Link Manager.<br><br>Updated requirements;<br>3.1.1.1.5 [VCC IP Prot: 0074/;-0]<br>3.1.1.4.1 [VCC IP Prot: 0080/;-1]<br>3.1.1.4.4 [VCC IP Prot: 0082/;-0]<br>3.1.1.4.5 [VCC IP Prot: 0083/;-0]<br>3.1.1.4.6 [VCC IP Prot: 0084/;-0]<br>3.1.1.5.1 [VCC IP Prot: 0087/;-0]<br>3.1.1.6.1 [VCC IP Prot: 0089/;-0]<br>3.1.1.6.3 [VCC IP Prot: 0091/;-0] | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| v1.8 DRAFT | 2013-10-10 | Correction:<br>3.1.1.1.4 [VCC IP Prot: 0073/;-0];<br>Since IP LM signals is not to be ASN.1 encoded make sure it is reflected in the VCC IP Command Protocol *DataType* header field as stated in 2.1.2.3.8 DataType | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| v1.8 DRAFT | 2013-10-11 | Clarification:<br><br>2.1.2.4.7OperationType NotificationRequest<br>Update to clearly state that initial notification messages are not to be sent on IP bus activation for signals specified to be dynamically requested with operation type NotificationRequest.<br>New requirements;<br>2.1.2.4.6.2 [VCC IP Prot: 0097/;-0]<br>2.1.2.4.7.2 [VCC IP Prot: 0098/;-0] | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| v1.8 REL | 2013-10-28 | eT: 10214828<br><br>3.1.1.4.2 [VCC IP Prot: 0099/;-0] : Faulty equation. Change Total_number_of_IP_nodes to | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |

| | | | |
|---|---|---|---|
| | | Number_of_Nodes_in_RG_X throughout the document | |
| v1.8 REL | 2013-10-28 | eT: 10216070<br><br>1.8 Template Definitions ; Added definitions<br><br>4.2.10 String (dynamic length) ; Added clarification.<br><br>3.1.1.1.5 [VCC IP Prot: 0074/;-0] ; Renaming of ACTION-names. Added reference to definition of RG.<br><br>3.1.1.4.1 [VCC IP Prot: 0080/;-1] ; Rename variable to Number_of_Nodes_in_RG_X.<br><br>3.1.1.4.2 [VCC IP Prot: 0099/;-0] ; Corrected formula<br><br>3.1.1.4.4[VCC IP Prot: 0082/;-0] ;<br>3.1.1.4.5[VCC IP Prot: 0083/;-0] ;<br>3.1.1.5.1[VCC IP Prot: 0087/;-0] ;<br>3.1.1.6.1[VCC IP Prot: 0089/;-0] ;<br>3.1.1.6.3[VCC IP Prot: 0091/;-0] ;<br>3.1.1.6.4[VCC IP Prot: 0092/;-0] ;<br>3.1.1.6.5[VCC IP Prot: 0093/;-0] ;<br>Updated variable names and terminology | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| v1.9 | 2013-11-22 | The following two requirements has been given the same requirements number, this is fixed and the later one is given number 0099.<br>2.1.2.3.8.1 [VCC IP Prot: 0094/;-0]<br>3.1.1.4.2 [VCC IP Prot: 0099/;-0] | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |
| v1.9 | 2013-11-22 | eT: 10218113<br>3.1.1.4.1 [VCC IP Prot: 0080/;-1] ; In the description CarConfig was mentioned by mistake. It is corrected. | 94141, Mohammad Mirzaei (mmirzaei), Volvo Car Corporation |

**Table 1: Document revision history**

**Note**: This is not the final version.

## 1.6 Volvo specification references

| Reg.nr./ID | Date | Description |
|---|---|---|
| 31835911 | | VCC Internetworks General Specification |
| 31836302 | | VCC IP Routing Specification |
| 31841985 | | VCC IP Command Bus – SPA |
| 31843680 | | VCC IP Command Bus - MCA |

**Table 2: Volvo specification references**

## 1.7 External Specification references

| Reg.nr./ID | Date | Description |
|---|---|---|
| ITU-T X.691 / ISO/IEC 8825-2 | | ASN.1 encoding rules: Specification of Packed Encoding Rules (PER) |

**Table 3: External specification references**

## 1.8 Template Definitions

| Definition | Description |
|---|---|
| Ack | Acknowledgement that a message has been received |
| ASN.1 | Abstract Syntax Notation One which is a standard defined by ISO and IEC and ITU-T. |
| Client | A software program which access services provided/enabled by the server |
| Offset | Indicates the distance from the beginning of an object up until a given point. Denotes the number of address location added to a base address in order to get a specific absolute address. |
| Operation | A function, method (or an instruction) |
| OperationID | Unique identifier of an operation, used to identify a function/method. |
| SenderHandleID | A unique identifier used to separate between the different and multiple calls to the same operation. |
| Service | A service is a subsystem, which is an independent component of a larger system. In this specification, it's a logical combination of methods or events. E.g. Telematics, WLAN … |
| ServiceID | Unique identifier of a service, used to identify a specific service. |
| Server | A software program that offers one or many services |
| Request | A message form the client to the server invoking a operation |
| Response | A message from the server to the client transporting the result of a operation invocation |
| Request/Response | A remote procedure call that consists of booth a request and a response. |
| Resource Group | Predefined group of vehicle internal IP ECUs used in VCC IP Link Manager |
| VCC IP Command Protocol | This specification, which defines how the communication shall be performed. |
| VCC IP Command Bus | VCC Specification which defines all messages (signal content) using the VCC IP protocol. |
| VCC IP Link Manager | Mechanism for managing, controlling and monitoring the vehicle internal IP links. |

**Table 4 : Definitions**

## 1.9 Template Abbreviations

| Abbr. | Stands for |
|---|---|
| API | Application Programming Interface |
| ARS | Active Request Session |
| BYTE | 8 bits, permitting values between 0 to 255 |
| DBC | Vector Informatik GmbH proprietary , File Extension Description |
| DLL | Dynamic-Link Library |
| FMEA | Failure Mode and Effects Analysis |
| IP | Internet Protocol |
| ISO | International Organization for Standardization |
| ITU-T | International Telecommunication Union – Telecommunication Standardization Sector ITU-T |
| LM | VCC IP Link Manager |
| LSC | Local Software Component |
| OSI | Open Systems Interconnection |
| PDU | Protocol Data Unit |
| RG | Resource Group |
| TCP | Transmission Control Protocol |
| TBD | To be defined |
| UDP | User Datagram Protocol |
| VCC | Volvo Car Corporation |
| WFA | Wait-For-Ack |
| WFR | Wait-For-Response |

**Table 5: AbbreviationsVCC IP COMMAND PROTOCOL**

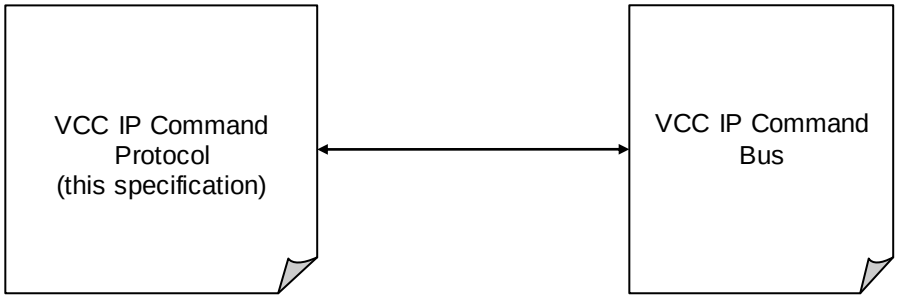
# 2. VCC IP COMMAND PROTOCOL

## 2.1 Introduction

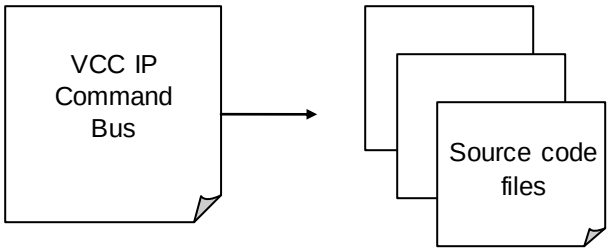
The VCC Internetwork specification [doc nr: 31835911] and VCC IP Routing specification [doc nr: 31836302] defines a set of rules that must be considered and fulfilled when designing/using the VCC IP Command Protocol and the control signals used by the applications (which is handled in separate documents).

**Design Rule**: If VCC applications (proprietary) intend to communicate internally within the vehicle network between IP based ECU's and when there is a need of a control channel, this specification (VCC IP Command Protocol) shall be used. The VCC IP Command Protocol specification provides requirements and guidelines how the data shall be exchanged between the ECU's. If the application under development is based on a well known protocol (standardized framework), this should be stated in the corresponding IP Signal Content specification (SWRS) and does not need to use the VCC IP Command Protocol). Example the usage of FTP (RFC-959) would mean that this specification is not needed.

VCC delivers a set of specification which shall be used together. The VCC IP Command Protocol which contains requirements and design guidelines which must be used for the implementations.
The VCC IP Command Bus which defines services which is offered by servers.




The VCC IP Command Bus generates a set of source-code files which shall be used for the implementations of the services.



The VCC IP Command Protocol defines controlling mechanisms which are suitable for IP-based (capable) networks (e.g. Ethernet STP, USB, APIX2 or MOST150). This means that the VCC IP Command Protocol is independent of the physical communication bus, as long as the communication bus has support for Ethernet. However, it could be possible to use the VCC IP Command Protocol over other automotive networks (e.g. Flexray), but this is not the main target for this specification.

The figure below shows how the communication is performed between the applications.

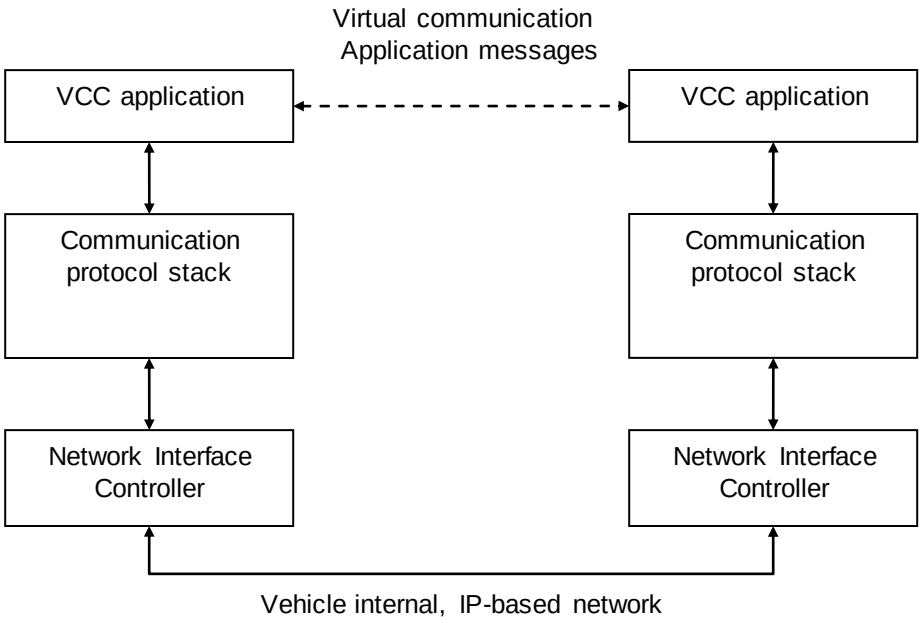



**Figure 1: Virtual communication between applications and real communication over the internal IP-based networks**

## 2.1.1 Transport Layer

The transport layer is responsible for the handling of IP addresses and port numbers to be used.
In this layer the terminology we use either client/sender or server/receiver.

- Client Address – Sender Address
- Client Port – Sender Port
- Server Address – Receiver Address
- Server Port – Receiver Port

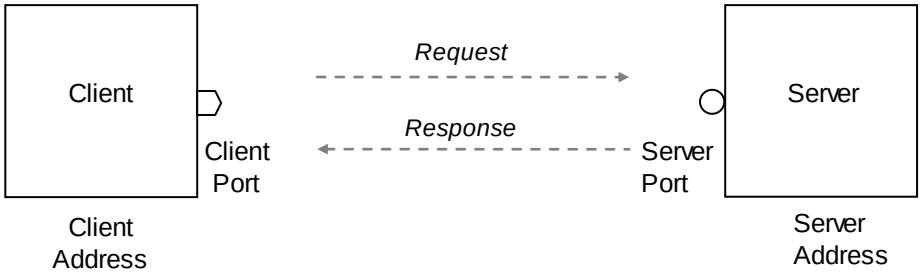



**Figure 2: Showing the relations on the transport layer**

The transport layer provides functionality to provide a reliable TCP or unreliable UDP data transfer service to upper layers.

The VCC IP Command Protocol supports both UDP and TCP. The usage of the transport protocol depends on the application use-case. Using UDP, the VCC IP Command Protocol defines some additional mechanism to handle acknowledgment messages, timeout and retransmissions which are explained in detail in this specification. If TCP is preferred, the VCC IP Command Protocol specification defines a set of rules on how the control data shall be used.

### 2.1.1.1 Generic reception handling of UDP/TCP data

The figure below shows a generic way to handle UDP/TCP data reception, and shall be read as an example. Application header and payload specific error handling is described in a more detailed form within other chapters of this specification. The figure below shows a generic flow on how to handle control messages. This specification will present further details on how to handle messages in other chapters.

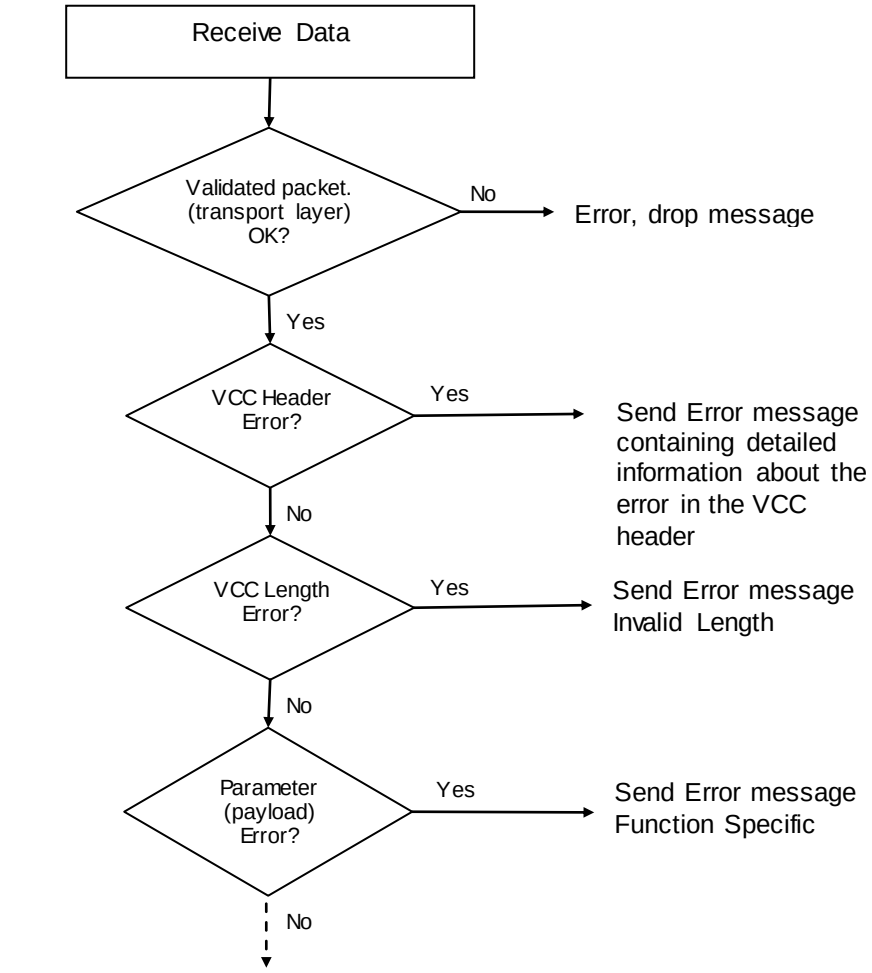



**Figure 3: Example, generic reception handling of VCC control messages using UDP/TCP data**

### 2.1.1.2 UDP

UDP uses a simple transmission, stateless and an unreliable service. Therefore data may arrive out of order, appear duplicated or go missing without a notice. It's suitable where the queries must be fast and only consist of a request followed by a reply packet. UDP can be used for hard latency requirements (<100 ms).

This specification presents how to use UDP for control messages in conjunction with the VCC IP Command Protocol and addresses how to handle retransmission and errors using UDP.

#### 2.1.1.2.1 Generic flowchart of UDP data

The figure below shows a generic flow when UDP is used between a client and server.

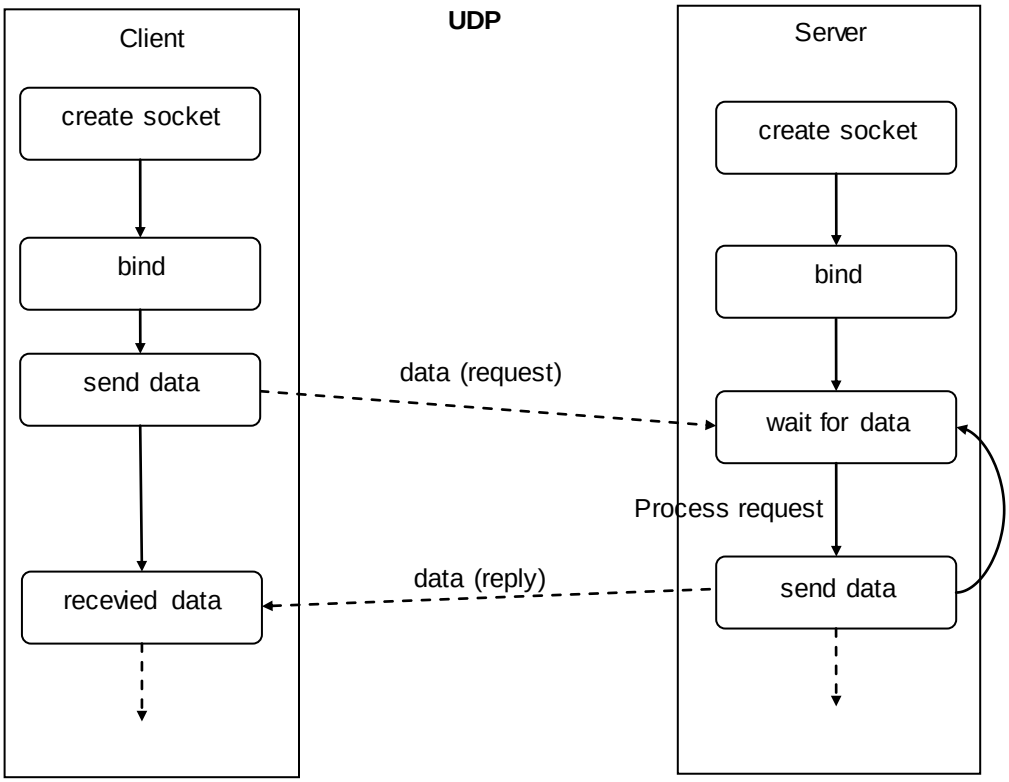



**Figure 4: Simplified UDP socket states**

More information about the VCC IP header structure can be found in 2.1.2.2.

## 2.1.1.2.2 VCC IP Command Protocol – General UDP requirements

### 2.1.1.2.2.1 [VCC IP Prot: 0001/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the application header for the control commands using UDP/IP. |
| Source | Chapter 2.1.2.2. |
| Verification Method | Test |

**Req:** The VCC IP Protocol header shall be used when transporting UDP packets for control and command between ECU's. The usage of UDP as transport layer shall be specified in the VCC IP Command Bus.

### 2.1.1.2.2.2 [VCC IP Prot: 0002/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define and clarify the usage of the UDP checksum for error-checking of the header and data. |
| Source | |
| Verification Method | Test |

**Req:** The UDP protocol shall be used according to VCC Internetwork specification [doc nr:31835911], where it's also stated that the usage of UDP checksum is mandatory.

### 2.1.1.2.2.3 [VCC IP Prot: 0003/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define when the UDP transport protocol shall be used for small control messages. |
| Source | |
| Verification Method | Test |

**Req:** UDP shall be used for small control messages (less or equal than 1400 bytes).

### 2.1.1.2.2.4 [VCC IP Prot: 0004/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define if segmentation shall be supported when the UDP transport protocol shall be used in conjunction with the VCC IP Protocol header. |
| Source | |
| Verification Method | Test |

**Req:** Using UDP, segmented messages shall not be used.

### 2.1.1.3 TCP

Using the VCC IP Protocol header, TCP shall be used for larger streams of data where reliable transmission is preferred since TCP supports (error detection, flow control, congestion control etc).

#### 2.1.1.3.1 Generic flowchart of TCP data

The figure below shows a generic flow when TCP is used between a client and server.

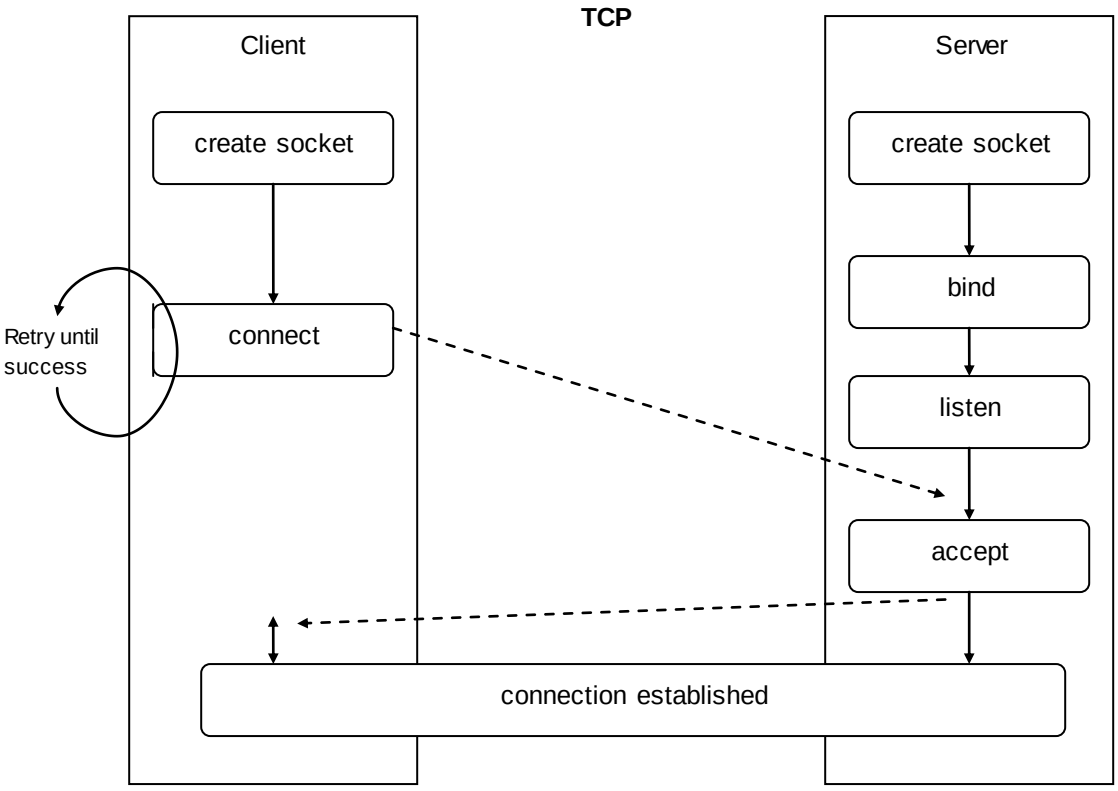



**Figure 5: Simplified TCP socket states from vehicle startup**

TCP allows usage of various algorithms to handle congestion and influence flow control. This information is specified according to VCC Internetworks General Specification [doc nr: 31835911].

More information about the header structure can be found in 2.1.2.2.

## 2.1.1.3.2 VCC IP Command Protocol – General TCP requirements

### 2.1.1.3.2.1 [VCC IP Prot: 0005/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the application header for the control commands using TCP/IP. |
| Source | Chapter 2.1.2.2. |
| Verification Method | Test |

**Req:** The VCC IP Command Protocol header shall be used when transporting TCP packets for control and command between ECU's. The usage of TCP as transport layer shall be clearly specified in the VCC IP Command Bus.

### 2.1.1.3.2.2 [VCC IP Prot: 0006/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define which transport protocol that should be used for segmented VCC IP Command Protocol messages. |
| Source | |
| Verification Method | Test |

**Req:** TCP shall be used for control messages larger than 1400 bytes.

### 2.1.1.3.2.3 [VCC IP Prot: 0007/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the TCP connection establishment and the startup behavior of the devices using the VCC IP Command Protocol... |
| Source | |
| Verification Method | Test |

**Req:** The IP based ECU's which shall use TCP as transport layer, shall establish a TCP connection from client to server at startup. If the client detects that the TCP connection is not available, it shall re-establish the TCP connection to the server.

Further details are explained in requirements: [VCC IP Prot: 0008/;-0], [VCC IP Prot: 0009/;-0] and [VCC IP Prot: 00010/;-0]

### 2.1.1.3.2.4 [VCC IP Prot: 0008/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the server behavior upon startup (power-on) when TCP is used. |
| Source | |
| Verification Method | Test |

**Req:** The server shall at startup initialize, open server port and listen on connection attempts.

### 2.1.1.3.2.5 [VCC IP Prot: 0009/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the client behavior upon startup (power-on), when TCP is used. |
| Source | |
| Verification Method | Test |

**Req:** The client shall at startup create a socket and try to connect to the server

### 2.1.1.3.2.6 [VCC IP Prot: 0010/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the server/client communication reestablishment mechanism upon startup (power-on), when TCP is used andconnection attempts fails. |
| Source | |
| Verification Method | Test |

**Req:** The client TCP connections re-establish attempts shall be performed every 100ms after a connection attempt has been refused.

## 2.1.1.4 Port numbers

A server application that offers a service running on UDP/TCP uses a unique port number for a specific service. It's recommend that application services use the same port number for UDP and TCP.

Note: For the VCC IP Command Protocol, the sender and receiver port shall be the same e.g. port 50000 as sender port and receiver port. The used port shall be specified in the signal database specifications (VCC IP Command Bus).

The IANA (Internet Assigned Numbers Authority) defines the following ranges:

- 0 to 1023 are defined as *well-known ports* and is widely-used within Internet
- 1024 to 49151 are registered and assigned ports by IANA
- 49152 to 65535 contains dynamic or private ports that may be used for custom or temporary purpose

### 2.1.1.4.1.1 [VCC IP Prot: 0011/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the port numbers to be used. |
| Source | IANA Ports, *Port Numbers, IANA,* *http://www.iana.org/assignments/port-numbers* |
| Verification Method | Test |

**Req:** The allowed communication port numbers shall be chosen according the IANA organization and clearly defined which ports shall be used within VCC specification which defines the signal content (VCC IP Command Bus).

## 2.1.2 Application Layer – VCC PDU

The following chapter defines a set of rules which shall be used when designing the control signals used by the applications. The VCC IP Command Protocol contains a PDU, which header shall be used by all IP based ECU's for control signaling.

### 2.1.2.1 Design overview

The functionalities of the services are offered as a set of operations For example, a service "A" or service "B" offers a set of operations as seen in the figure below. Each service shall have a unique identifier and each service operation shall be assigned its unique identifier to distinguish them from one another.

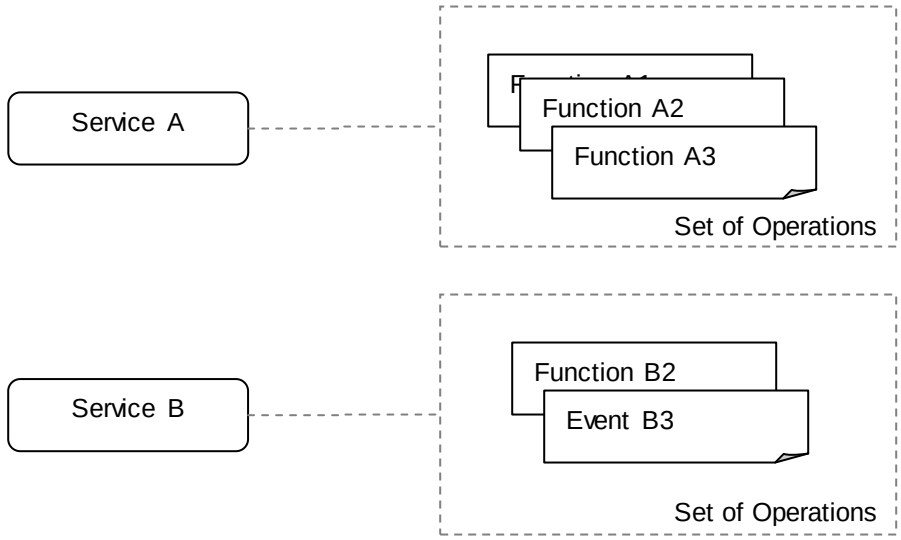



**Figure 6: Showing relations between services and operations**

This specification does not intend to specify any details on how to protect sensitive data that needs of protection. Applications shall use security protocols which prevents external resources to read the specific network communication.

More detailed information about the VCC PDU can be found in the next subchapters.

### 2.1.2.2 Header Overview

For VCC IP Command Protocol communication, the figure below shows the relation between the VCC PDU header on the application layer, the UDP/TCP handling on the transport layer and also the IP handling on the network layer.

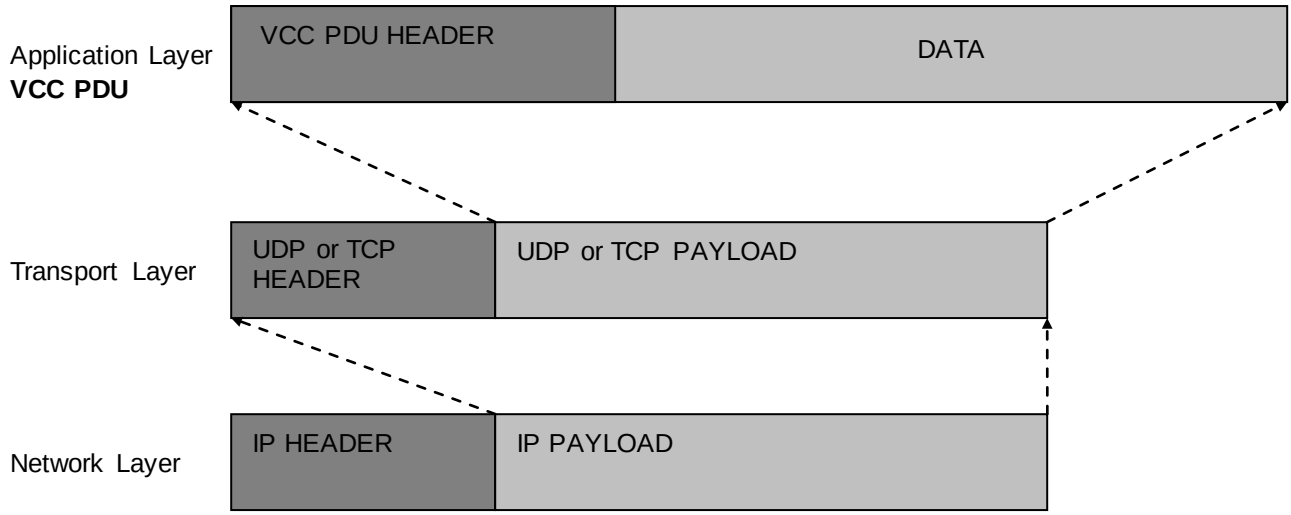



**Figure 7: Overview figure of VCC (PDU) proprietary data structure using either UDP or TCP**

### 2.1.2.3 Header Structure

The VCC PDU header consists of 8 mandatory fields according to the figure below. The data field is variable and optional.

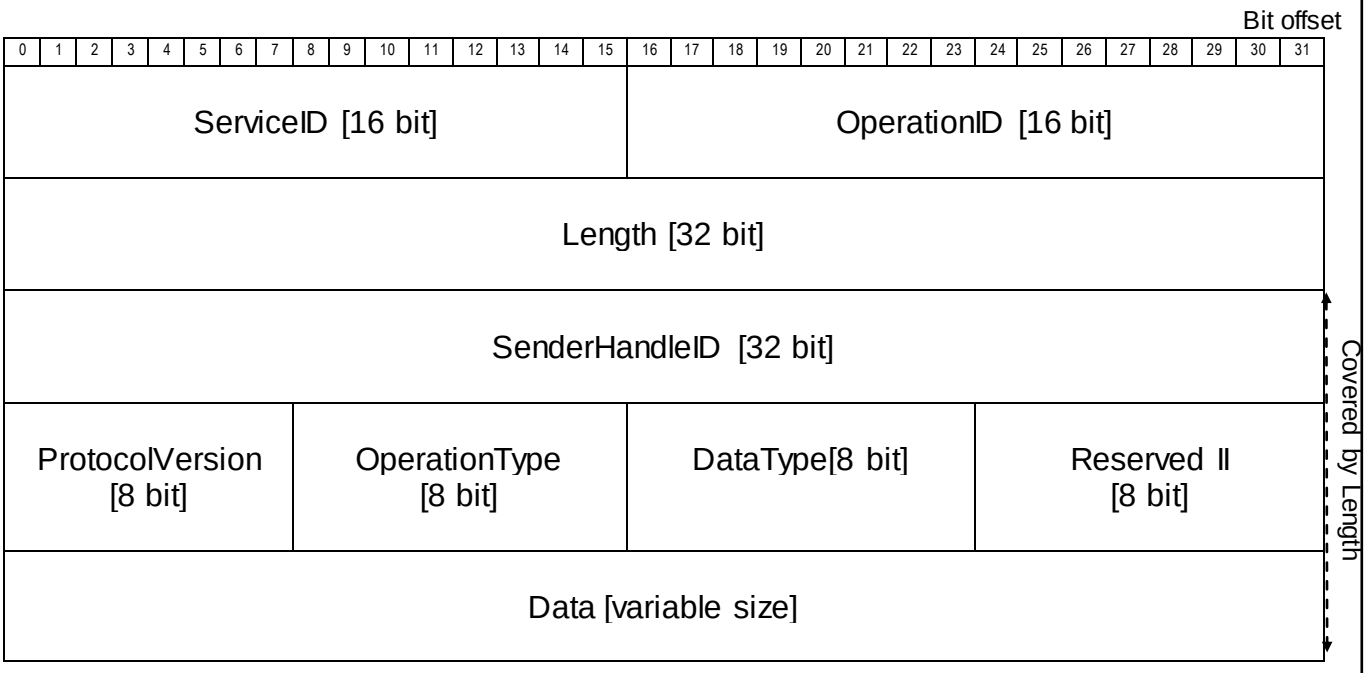



**Figure 8: VCC IP Header Format on Application Layer**

### 2.1.2.3.1 Overview of the VCC IP Command Protocol header format

**ServiceID:** 16 bit (0x0000 – 0xFFFF)
**OperationID:** 16 bit (0x0000 – 0xFFFF)
**Length:** 32 bit (0x00000008 – 0xFFFFFFFF)
**SenderHandleID:** 32 bit (0x00000000 – 0xFFFFFFFF)
**ProtocolVersion:** 8 bit (0x00 – 0xFF)
**OperationType:** 8 bit (0x00 – 0xFF)
**DataType:** 8 bit (0x00 – 0xFF)
**ReservedII**: 8 bit (0x00 – 0xFF)
**Data:** variable size

VCC IP Application Header Size: 16 bytes (128 bits) in total.

#### 2.1.2.3.1.1 [VCC IP Prot: 0012/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define when the VCC IP Command Protocol header shall be used. |
| Source | |
| Verification Method | Analysis |

**Req:** The VCC IP Command Protocol application header format must be used for control and command communication within the vehicle IP-based network and all headers must be encoded with network byte order (big endian) [RFC 791].

### 2.1.2.3.2 ServiceID

The 16 bit service identifier (ServiceID) is used to identify a specific service that a server provides. Example of services provided by the VCC subsystems are Telematics, Phone, Connectivity etc.

#### 2.1.2.3.2.1 [VCC IP Prot: 0013/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define where the unique service identifiers shall be listed. |
| Source | |
| Verification Method | Analysis |

**Req:** The `ServiceID` is allocated by document owner and shall be listed into the VCC signal specifications (VCC IP Command Bus).

### 2.1.2.3.3 OperationID

The 16 bit `OperationID` shall be used to identify which service operation that shall be used. The `OperationID` must be unique for each operation that is supported in the vehicle.

#### 2.1.2.3.3.1 [VCC IP Prot: 0014/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define where the unique OperationID identifiers shall be listed. |
| Source | |
| Verification Method | Analysis |

**Req:** The `OperationID` is allocated by document owner and shall be listed into the VCC signal specifications (VCC IP Command Bus).

### 2.1.2.3.4 Length

The Length-field consists of 32 bits containing the length in Byte of the payload. The length shall also include the following parameters into the length calculation; the `SenderHandleID`, `ProtocolVersion`, `OperationType`, `DataType` and `Reserved II`, which means that the minimum length shall always be 8 bytes.

**Note**: The beginning of the length calculation is at the beginning of the `SenderHandleID`, offset 64.

#### 2.1.2.3.4.1 [VCC IP Prot: 0015/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define that length field is allways mandatory |
| Source | |
| Verification Method | Analysis |

**Req:** The `Length` parameter is mandatory and shall be included in the VCC IP application messages.

### 2.1.2.3.5 SenderHandleID

With UDP there is a lack of flow control, to be able to identify messages SenderHandleID shall be used.

SenderHandleID is a locally (in the application) unique "message session" identifier used to tell apart multiple replies from the same source It allows a client do differentiate multiple calls to the same operation.

**SenderHandleID [32 bits]**

| ServiceID8 [8 bits] | OperationID8 [8 bits] | OpType [8 bits] | SeqNr [8 bits] |
|---|---|---|---|

The picture above illustrates the way the SenderHandleID shall be constructed to fulfill the required behavior.
ServiceID8 shall consist of the last byte of the 16 bit ServiceID.
OperationID8 shall consist of the last byte of the 16 bit OperationID.
OpType shall use the same value as the OpType.
SeqNr is a sequence number shall be locally unique in the client application for each message to be sent.

Note: The SenderHandleID field may change in the future.

More information regarding usage of SenderHandleID and timers in chapter: Timers

#### 2.1.2.3.5.1 [VCC IP Prot: 0016/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the SenderHandleID shall be used. |
| Source | |
| Verification Method | Analysis |

**Req:** The `SenderHandleID` is mandatory and must be constructed as described in chapter 2.1.2.3.5 .

#### 2.1.2.3.5.2 [VCC IP Prot: 0017/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the SenderHandleID shall be used. |
| Source | |
| Verification Method | Analysis |

**Req:** When generating a response message, after a request message the server shall copy the `SenderHandleID` from the request message to the response message.

The `SenderHandleID` may be reused as soon as the response arrived or is not expected to arrive anymore (timeout).

### 2.1.2.3.5.3 [VCC IP Prot: 0018/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the applications shall use SenderHandleID to distinguish between different application messages. |
| Source | |
| Verification Method | Analysis |

**Req:** The applications shall use the `SenderHandleID` and the IP-source address to distinguish between different messages.

### 2.1.2.3.6 ProtocolVersion

An identifier of the version of the VCC IP Command protocol which is used.

### 2.1.2.3.6.1 [VCC IP Prot: 0019/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the current version number of the VCC IP Command Protocol. |
| Source | |
| Verification Method | Analysis |

**Req:** The currently used VCC IP Command Protocol number: 0x02

### 2.1.2.3.7 OperationType

Identifier to differentiate between different types of messages, request, notifications, response etc. Each OperationType will be explained further in detail in the following subchapters in conjunction with Message Format Sequences.

#### 2.1.2.3.7.1 [VCC IP Prot: 0020/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the operationtypes that shall be used for the VCC IP Command Protocol. |
| Source | |
| Verification Method | Analysis |

**Req:** The `OperationTypes` according to Table 6: Presenting valid Operation Types.

| Number | Friendly Name | Description |
|---|---|---|
| 0x00 | `REQUEST` | Request where a response is necessary.<br>E.g. The client uses a `REQUEST` to get the value of a property from the server. The value is returned to the client in a `RESPONSE` message. |
| 0x01 | `SETREQUEST_NORETURN` | A message that does not require any response from the receiver. Using UDP, this OperationType is only used in conjunction with notification messages. E.g. set a new value. |
| 0x02 | `SETREQUEST` | A combined message that performs two operations into one namely Set and Get. First, the client sets a new value and waits for the response consisting of the updated value. |
| 0x03 | `NOTIFICATION_REQUEST` | A request of a notification/event callback.<br>Used to request to subscribe on a property, either cyclic or event based. |
| 0x04 | `RESPONSE` | The `RESPONSE` message for a `REQUEST`. E.g. the status message containing the current value. |
| 0x05 | `NOTIFICATION` | Notifications are typically used when an application subscribes on a certain parameter which is sent on events for this notification type. The receiver of a message of this notification type shall respond with an ACK message. |
| 0x06 | `NOTIFICATION_CYCLIC` | This notification type is sent cyclically according to the requested period, and shall not be acknowledged (that is NOT responded with an ACK) by the received. |
| 0x07…0x6F | - | Reserved by IP API Template |
| 0x70 | `ACK` | Acknowledgement for UDP messages.<br><br>The sender/receiver shall make use of the SenderHandleID (and the IP source address) to identify/distinguish which message to send/receive acknowledgement on.<br>Note: The acknowledgment messages do not carry any payload. |
| 0xE0 | `ERROR` | This message type is a response containing an error message. The first payload byte, consists of the 8 bit `ERRORCODE` followed by 16 bit `ERRORINFORMATION`.<br>More information regarding general error codes can be found in chapter 2.1.4 |

**Table 6: Presenting valid Operation Types**

Further examples on how to use the `OperationTypes` is be presented in chapter 2.1.2.4.

### 2.1.2.3.8 DataType

### 2.1.2.3.8.1 [VCC IP Prot: 0094/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the usage of `DataType` |
| Source | |
| Verification Method | Test |

**Req:** The `DataType` uses 16 bits which purpose is to define how the sender/receiver shall handle the payload (data field) of the message.

The table below defines the value of the parameter:

| FriendlyName: | Value: | Description: |
|---|---|---|
| Encoded message | 0x00 | Indicates that the payload (data) is encoded according to requirement [VCC IP Prot: 0022/;x]. |
| Normal message | 0x01 | The payload (data) is send without any encoding. The data is packed in byte order or as defined in corresponding SWRS which defines how the data shall be packed. |

**Table 7: Defining the DataType parameter**

### 2.1.2.3.9 Reserved II:

16 bit area, reserved for further usage, set to 0x00.

### 2.1.2.3.10 Data

### 2.1.2.3.10.1 [VCC IP Prot: 0021/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the data field is encoded. |
| Source | |
| Verification Method | Test |

**Req:** The data field (payload) must be encoded within network byte order (big endian) [RFC 791]. The data shall be defined in the VCC IP Command Bus specification.

### 2.1.2.3.10.2 [VCC IP Prot: 0022/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the data field is encoded. |
| Source | Chapter 4 Appendix A - asn.1 EXAMPLES |
| Verification Method | Test |

**Req:** If the message shall be encoded according to the `DataType 0x00` [VCC IP Prot: 0094/;-x], the Abstract Syntax Notation One (ASN.1) with PER-unaligned shall be used. The default character encoding shall be UTF-8, used by all ASN.1 strings.

Details of how ASN.1 shall be used is explained in Appendix A - asn.1 EXAMPLES.

## 2.1.2.4 Application Message Format Sequences

This chapter defines the client/server communication within a service, using different `OperationTypes`. The message format sequences are divided in subchapter which explains the differences between the usage of UDP or TCP as transport protocol.

When client creates a new control message using the VCC IP Command Protocol, the following data will be transferred between the client and server.

Client: -> ServiceID.OperationID.Length.SenderHandleID.ProtocolVersion.OperationType.ReserverdI.ReservedII.data

**Note:** Due to the limitation of drawing space in this specification, the complete content of the application message is too long to present in the message format sequence figures. Therefore, only the `OperationID`, `OperationType` will be presented in the figures in this specification.

Client: -> ServiceID.**OperationID**.Length.SenderHandleID.ProtocolVersion.**OperationType**.ReserverdI.ReservedII.data

The `OperationType` will be replaced with the Friendly Name according to Table 6: Presenting valid Operation Types.

Further details on how the client and the server will handle the messages is described in chapter: 2.1.3.

### 2.1.2.4.1 Transport Protocol Requirements

#### 2.1.2.4.1.1 [VCC IP Prot: 0023/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define when the acknowledgment messages shall be used, using the VCC IP Command Protocol using UDP. |
| Source | |
| Verification Method | Test |

**Req:** When using UDP it's required that acknowledgment (ACK) messages are used for all messages (independent of OperationType).

The `ACK` message is defined as a separate message (operation type) within the requirement: [VCC IP Prot: 0020/;-0]

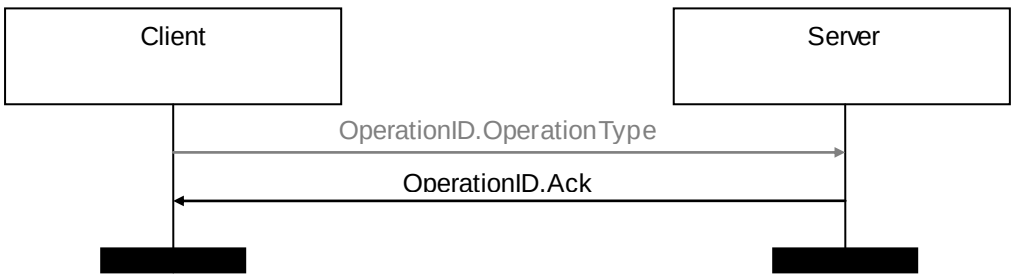



**Figure 9: Showing usage of ACK messages**

### 2.1.2.4.1.2 [VCC IP Prot: 0024/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define when the error messages shall be used, using the VCC IP Command Protocol and UDP/TCP. |
| Source | |
| Verification Method | Test |

**Req:** When using UDP or TCP, a sanity check must be done upon reception to detect errors in the message header. If a faulty message is received, the receiver is responsible to send back a message with OperationType: `ERROR` and relevant error code.

The `ERROR` message is defined as a separate message (operation type) within the requirement: [VCC IP Prot: 0020/;-0]. The reason for errors could be according to the general error codes or operation specific error codes, specified in the subchapter: 2.1.4.

If UDP is used and if the server detects a faulty control message, the server will respond with an `ERROR` message instead of an `ACK` message, as exemplified in Figure 10. The client shall interpret the `ERROR` message as an `ACK` message on the "faulty control message".

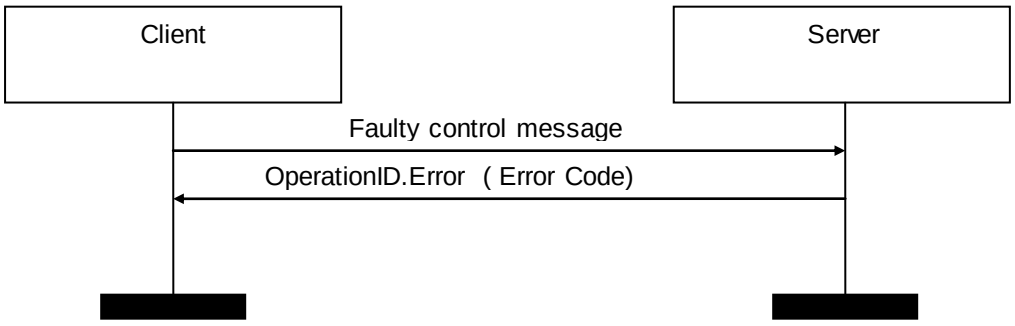



**Figure 10: Alternative message sequence of ERROR messages using UDP**

### 2.1.2.4.1.3 [VCC IP Prot: 0025/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the server (receiver) shall handle a request, when either UDP or TCP is used. |
| Source | |
| Verification Method | Test |

**Req:** On the server application, upon reception of a `SETREQUEST` message, the message content (incase of request of a new value, e.g. a new setting) shall be first stored in persistent memory before answering with a `RESPONSE` message.

### 2.1.2.4.1.4 [VCC IP Prot: 0026/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define timer handling (message handling) on the client and server side using UDP as transport protocol. |
| Source | |
| Verification Method | Test |

**Req:** When UDP is used as transport protocol, there shall be two timers:

- "wait-for-ack", WFA
- "wait-for-response" WFR

When TCP is used, there shall be one timer:

- "wait-for-response" WFR

Both WFA and WFR are responsible for guaranteeing that control messages are delivered within a certain time. More information about the timers can be found in chapter 2.1.3.1.

### 2.1.2.4.2 OperationType Request/Response

#### 2.1.2.4.2.1 UDP

The following section explains the details when UDP is used as transport protocol for OperationType `REQUEST` and `RESPONSE`.

##### 2.1.2.4.2.1.1 [VCC IP Prot: 0027/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the message sequence for a request/response when UDP is used. |
| Source | |
| Verification Method | Test |

**Req:** The `REQUEST /RESPONSE` shall be implemented according to the *Figure 11: Message Sequence Format – OperationType Request/Response using UDP* which shows the message sequence format when UDP is used in conjunction with the VCC IP Command Protocol header.

In an optional scenario (explained in figure Figure 12) where an error occurs in the `REQUEST`, the server is responsible for sending an error message using OperationType `ERROR`, instead of sending the `RESPONSE` message. The reason for errors could be according to the general error codes specified in the subchapter: 2.1.4.

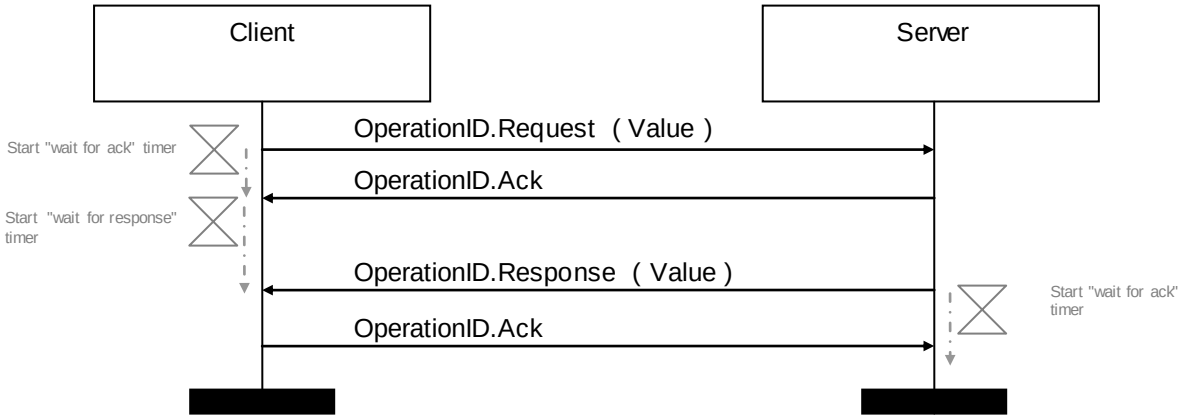



**Figure 11: Message Sequence Format – OperationType Request/Response using UDP**

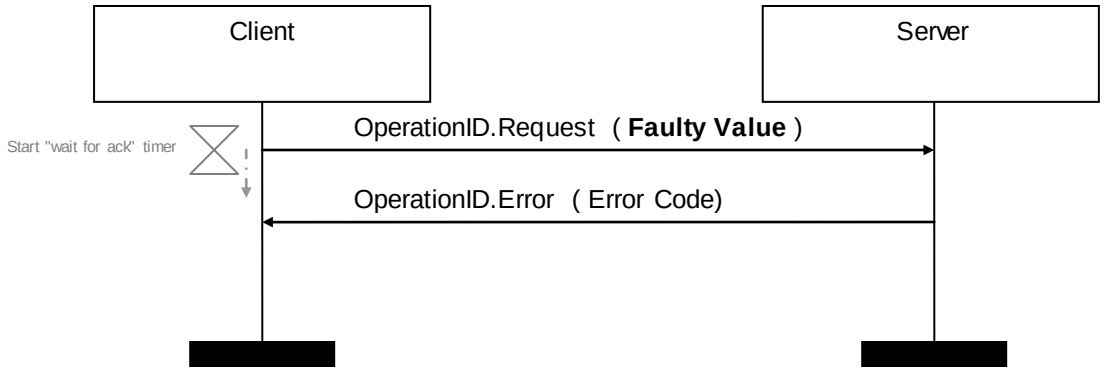



**Figure 12: Message Sequence Format – Alternative Request sequence with an Error using UDP**

### 2.1.2.4.2.1.2 [VCC IP Prot: 0028/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define acknowledgment handling on the sender and receiver side using UDP. |
| Source | Figure 11: Message Sequence Format – OperationType Request/Response using UDP |
| Verification Method | Test |

**Req:** When a REQUEST is made by a client, the server shall send an acknowledgment (ACK) message on the REQUEST (after validation check), before it processes the RESPONSE message.

When the server sends the RESPONSE message to the client and it shall start waiting for an ACK message from the client, to make sure that the RESPONSE message is delivered.

**Note:** If the client sends a REQUEST and receives a correct (expected) RESPONSE prior the expected ACK message, the RESPONSE message shall be handled as correct ACK message. The client shall then respond with an ACK message on the RESPONSE message.

Further details on how the client and the server will handle the messages is described in chapter: 2.1.3

### 2.1.2.4.2.2 TCP

The following section explains the details when TCP is used as transport protocol for OperationType `REQUEST` and `RESPONSE`.

#### 2.1.2.4.2.2.1 [VCC IP Prot: 0029/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the message flow using TCP. |
| Source | |
| Verification Method | Test |

**Req:** The message exchange between a client and a server using TCP, the VCC IP Command Protocol and the `REQEUST` and `RESPONSE` message shall be implemented according to *Figure 13: Message Sequence Format – OperationType Request/Response using TCP*.

In an optional scenario where an error occurs in the `REQUEST`, the server is responsible for sending an error message using OperationType `ERROR`. The reason for errors could be according to the general error codes specified in the subchapter: 2.1.4.

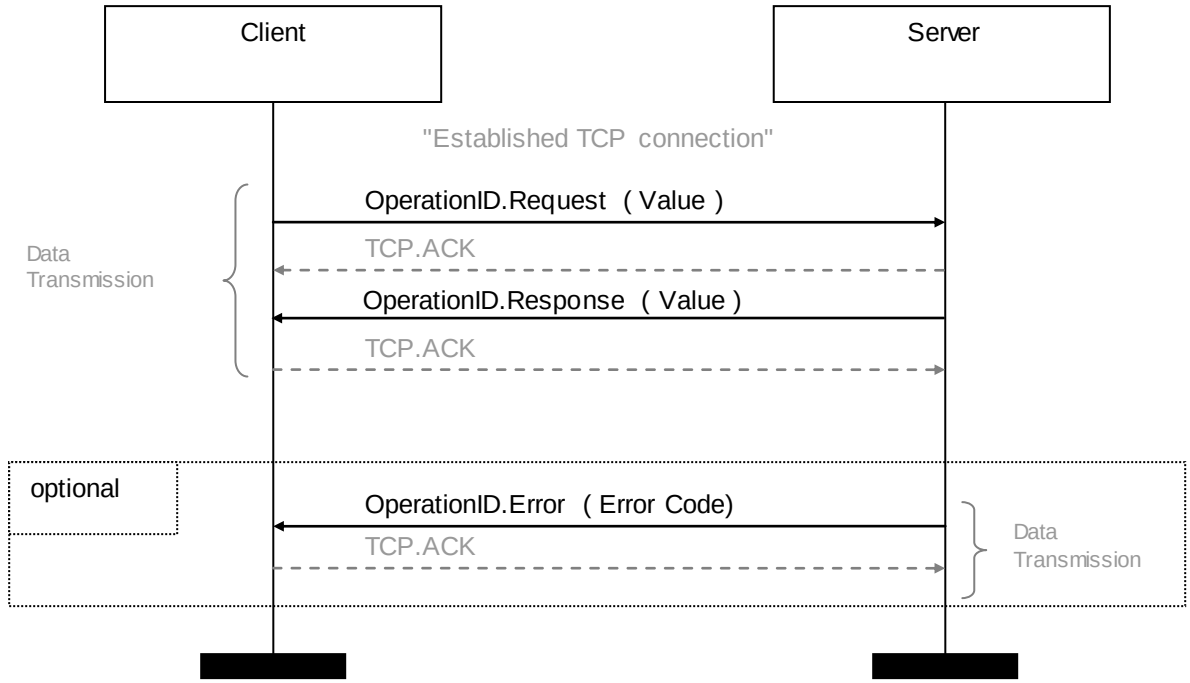



**Figure 13: Message Sequence Format – OperationType Request/Response using TCP**

### 2.1.2.4.2.3 Design guidelines

The following chapters shall be handled as a guideline on how a client and server creates REQUEST/RESPONSE messages correctly. Further details on how the client and the server will handle the messages is described in chapter: 2.1.3

**1:**
The client creates a new message and fills the header with new content:

- Construct the payload
- Set the ServiceID and the OperationID
- Set the Length field to 8 bytes + length of the rest of the header (8 bytes) + the payload
- Set a new unique SenderHandleID
- Set the ProtocolVersion
- Set the OperationType to REQUEST
- Set the DataType to: 0x00 (if the payload of the message shall be encoded)
- Set the ReservedII to: 0x00

**2:**
When the server receives the REQUEST, it's responsible to check the incoming message for errors and build up the new header based on the client header. It shall send an ACK message according to the following:

- Empty payload
- Copy the incoming ServiceID and OperationID to the outgoing message.
- Set the Length field to 8 bytes + length of the rest of the header (8 bytes)
- Copy the incoming SenderHandleID to the outgoing message
- Set the ProtocolVersion
- Set the OperationType to ACK
- Set the DataType to: 0x00 (if the payload of the message shall be encoded)
- Set the ReservedII to: 0x00

When the ACK message has been sent, the server shall create the RESPONSE message according to the following:

- Construct the payload
- Set the ServiceID and the OperationID
- Set the Length field to 8 bytes + length of the rest of the header (8 bytes) + the payload
- Copy the incoming SenderHandleID to the outgoing message
- Set the ProtocolVersion
- Set the OperationType to RESPONSE
- Set the DataType to: 0x00 (if the payload of the message shall be encoded)
- Set the ReservedII to: 0x00

**3:**
Last action on the client, is to send an ACK message on the received RESPONSE message:

- Empty payload
- Copy the incoming ServiceID and OperationID to the outgoing message.
- Set the Length field to 8 bytes + length of the rest of the header (8 bytes)
- Copy the incoming SenderHandleID to the outgoing message
- Set the ProtocolVersion
- Set the OperationType to ACK
- Set the DataType to: 0x00 (if the payload of the message shall be encoded)
- Set the ReservedII to: 0x00

### 2.1.2.4.3 OperationType SetRequestNoReturn

If a client wants' to set a new value and doesn't care about the result, the operation type: SETREQUEST_NORETURN shall be used.

#### 2.1.2.4.3.1 UDP

The following section explains the details when UDP is used as transport protocol for OperationType SETREQUEST_NORETURN.

##### 2.1.2.4.3.1.1 [VCC IP Prot: 0030/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the message flow using UDP for SETREQUEST_NORETURN |
| Source | |
| Verification Method | Test |

**Req:** When using UDP and sending a SETREQUEST_NORETURN, the client shall wait for the ACK message. If no ACK message is received, the client resends the SETREQUEST_NORETURN message.

**Note:** When using UDP, SETREQUEST_NORETURN shall only be used in conjunction with event based notification messages. Example, if a client tries to set an invalid (or valid) value the client will have the feedback on the following notification message from the server.

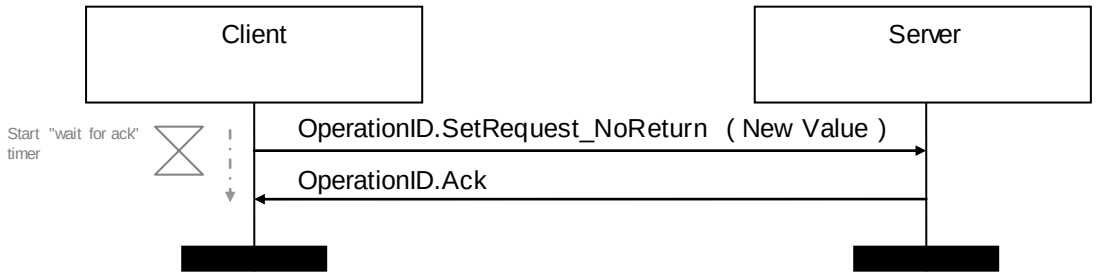



**Figure 14: Message Sequence Format – OperationType SetRequestNoReturn using UDP**

In case of the faulty message is detected at the server side, an ERROR message shall be send instead of the ACK message. The received ERROR message shall be interpreted by the client as a correct ACK message.

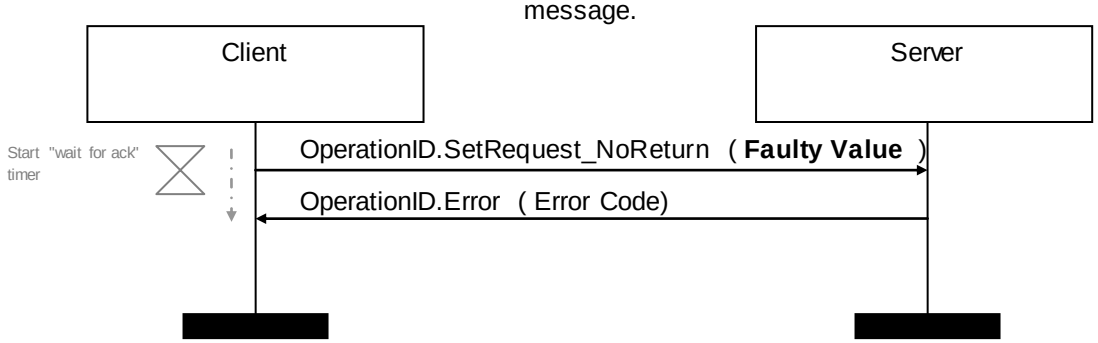



**Figure 15: Message Sequence Format - Error occurs using UDP and OperationType SetRequestNoReturn**

### 2.1.2.4.3.2 TCP

The following section explains the details when TCP is used as transport protocol for OperationType SETREQUEST_NORETURN.

#### 2.1.2.4.3.2.1 [VCC IP Prot: 0031/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the message flow using TCP for SETREQUEST_NORETURN. |
| Source | |
| Verification Method | Test |

**Req:** When using TCP and sending a SETREQUEST_NORETURN, the *Figure 16: Message Sequence Format – OperationType SetRequestNoReturn using TCP* shall be used.

If the server detects an error in the received message, it shall generate and send back an ERROR message which explains the reason for the error.

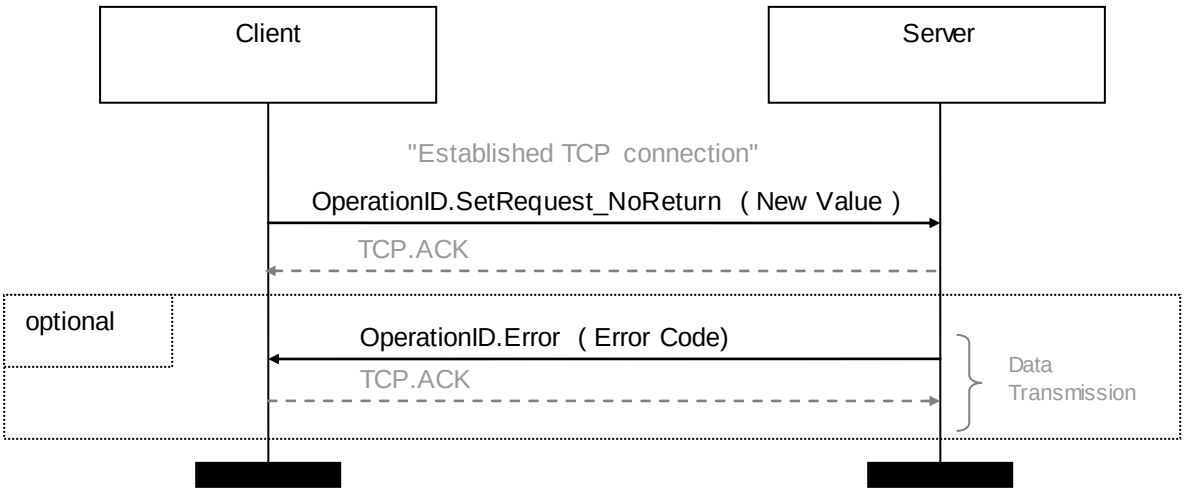



**Figure 16: Message Sequence Format – OperationType SetRequestNoReturn using TCP**

### 2.1.2.4.4 OperationType SetRequest

If a client/sender first wants to set a new value and then read the latest value (to confirm that the request is performed by the receiver, e.g. to make sure that the new value is written into the memory of the receiver), the OperationType SETREQUEST shall be used.

#### 2.1.2.4.4.1 UDP

The following section explains the details when UDP is used as transport protocol for OperationType SETREQUEST.

##### 2.1.2.4.4.1.1 [VCC IP Prot: 0032/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the message flow using UDP for SETREQUEST. |
| Source | |
| Verification Method | Test |

**Req:** When SETREQUEST is used, a reply of type ACK is expected prior a RESPONSE message (if no error occurs in the SETREQUEST). The SETREQUEST shall be implemented according to *Figure 17: Message Sequence Format – OperationType SetRequest using UDP*.

**Note:** If the client sends a SETREQUEST and receives a correct (expected) RESPONSE prior the expected ACK message, the RESPONSE message shall be handled as correct ACK message. The client shall then respond with an ACK message on the RESPONSE message.

If the health check has shown that there are errors in the SETREQUEST message, the server shall send an ERROR message to the client containing relevant error code instead of the ACK or RESPONSE message. Further details are explained in figure *Message Sequence Format – Alternative sequence with an Error using UDP.*

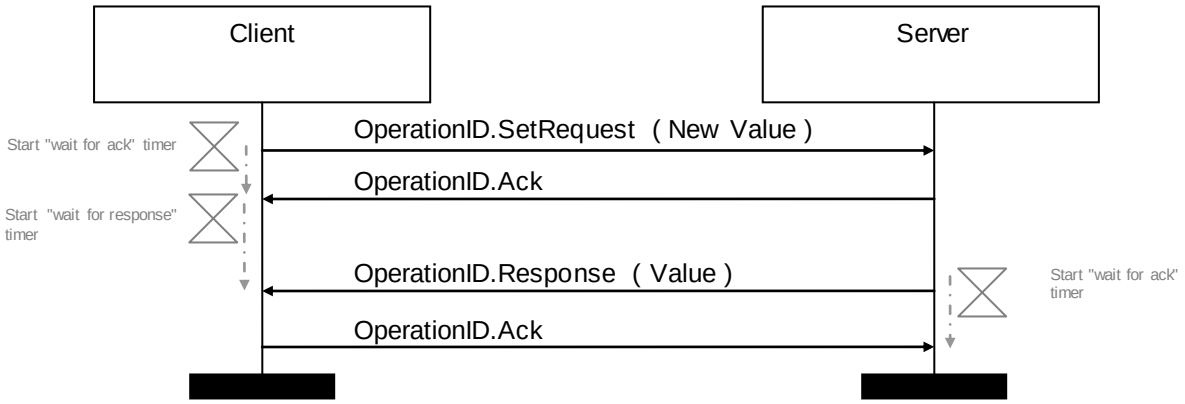



**Figure 17: Message Sequence Format – OperationType SetRequest using UDP**

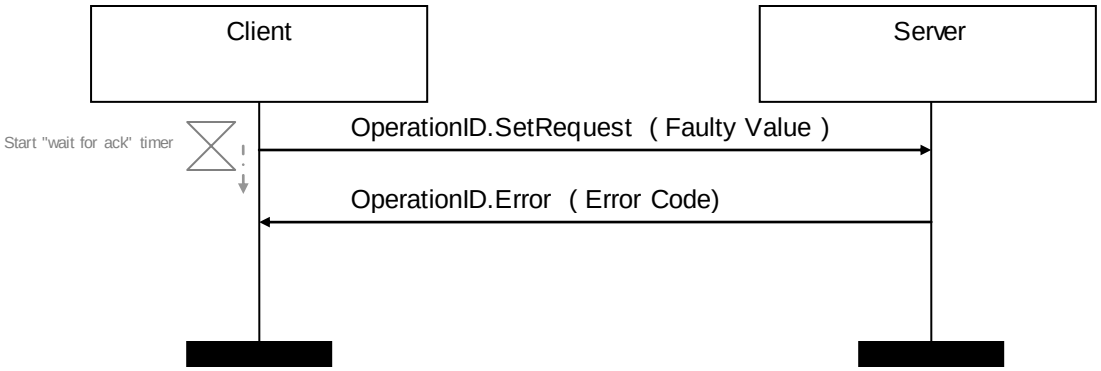



**Figure 18: Message Sequence Format – Alternative sequence with an Error using UDP**

### 2.1.2.4.4.2 TCP

The following section explains the details when TCP is used as transport protocol for OperationType SETREQUEST.

#### 2.1.2.4.4.2.1 [VCC IP Prot: 0033/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the message flow using TCP for SETREQUEST. |
| Source | |
| Verification Method | Test |

**Req:** When SETREQUEST is used, a reply of type RESPONSE (if no syntax error occurs in the SETREQUEST) is expected. If the health check has shown that there are errors in the SETREQUEST message, the server shall send an ERROR message to the client containing relevant error code, instead of the RESPONSE message.

The SETREQUEST shall be implemented according to *Figure 19: Message Sequence Format – OperationType SetRequest using TCP.*

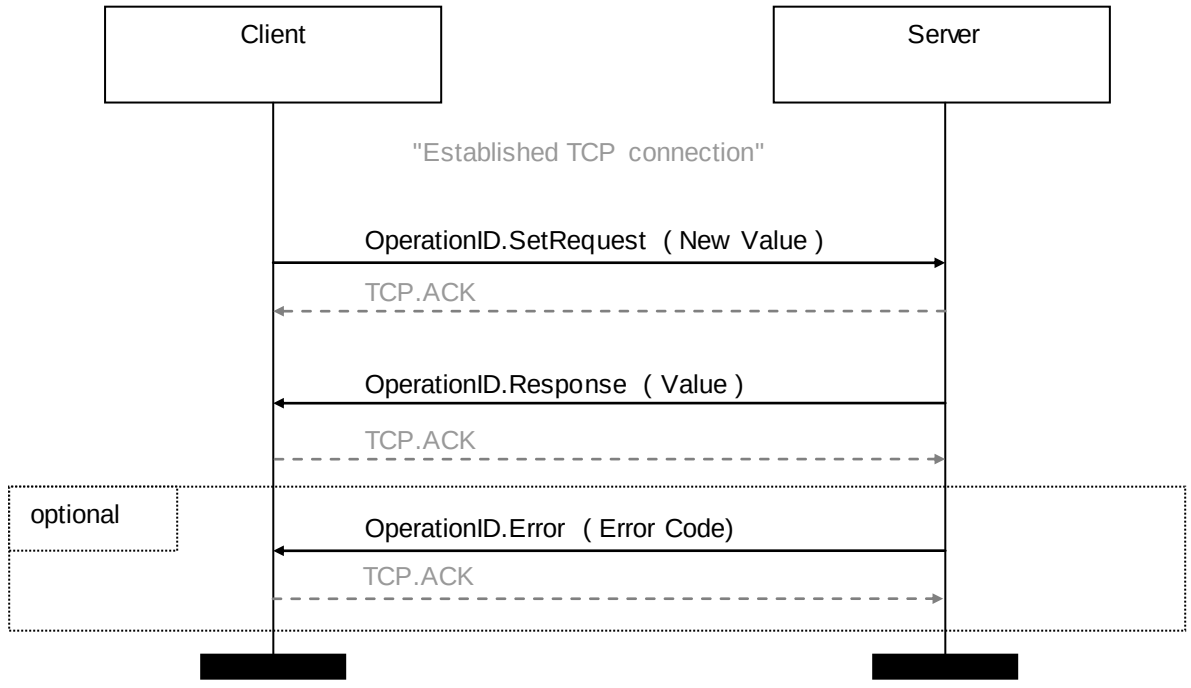



**Figure 19: Message Sequence Format – OperationType SetRequest using TCP**

### 2.1.2.4.5 Notification message Requirements

In some cases, it might be suitable to use notifications automatically sent on events, which is when a property has been changed. This it to avoid having processes consuming resources by continually polling for updates. Another type of notification is the cyclic notifications, where specified information shall be transmitted to the client in a fixed interval.

The way to handle notifications can either be implemented as static notifications (hardcoded) or dynamic where a device requests to receive notifications on certain properties.

#### 2.1.2.4.5.1 [VCC IP Prot: 0034/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define two different type of notifications. |
| Source | [VCC IP Prot: 0020/;-1] |
| Verification Method | Test |

**Req:** There shall be two types of notifications namely `NOTIFICATION and NOTIFICATION_CYCLIC`.

- `NOTIFICATION` (event-based notification messages sent upon "value" changes e.g. button pushed)
- `NOTIFICATION_CYCLIC` (messages sent based on fixed interval e.g. every 100ms)

#### 2.1.2.4.5.2 [VCC IP Prot: 0035/;-2]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define what shall be included in notification messages. |
| Source | |
| Verification Method | Test |

**Req:** A `NOTIFICATION` message shall be sent for every event and contain the new value set at that moment, when using event based notifications. When using `NOTIFICATION_CYCLIC` the message shall contain the latest values.

#### 2.1.2.4.5.3 [VCC IP Prot: 0036/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how NOTIFICATION_REQUEST shall be implemented. |
| Source | |
| Verification Method | Test |

**Req:** Operation type `NOTIFICATION_REQUEST` is used when an application (in runtime) requests to subscribe on a property provided by a service. If the request is sent as a UDP packet, the application shall wait for the `ACK` before expecting to receive any notification messages from the service, containing the latest property value.

### 2.1.2.4.5.4 [VCC IP Prot: 0037/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how notifications shall be implemented. |
| Source | |
| Verification Method | Test |

**Req:** Notification messages (no matter which type) shall never be followed up with an `ERROR` message.

### 2.1.2.4.6 OperationType Notification and Cyclic Notification

In case of static (pre-configured) notifications is preferred, the service shall implement a mechanism to distribute notifications to the interested IP based devices. It is up then to the server to inform which nodes that shall receive certain notification messages. The server (reporter) pushes data to the predefined clients (subscribers).

### 2.1.2.4.6.1 [VCC IP Prot: 0038/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how notifications shall be implemented using UDP or TCP. |
| Source | |
| Verification Method | Test |

**Req:** Notification messages shall be setup/started immediately after the link is established, to the subscribers.

- For type `NOTIFICATION`, the latest/current value shall be sent
- For type `NOTIFICATION_CYCLIC`, transmission shall be done according to the chosen timer-intervals.

### 2.1.2.4.6.2 [VCC IP Prot: 0097/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how notifications shall be implemented using UDP or TCP in relation to operation type NOTIFICATION_REQUEST. |
| Source | |
| Verification Method | Test |

**Req:** When an application is subscribing for notifications dynamically (no matter for which notification type) an initial notification message shall not be sent when “link up” as described for static notifications (as in requirement [VCC IP Prot: 0039/;-1]). In case the application still wishes to receive notifications, it shall send a new NOTIFICATION_REQUEST.

### 2.1.2.4.6.3 UDP

The following section explains the details when UDP is used as transport protocol for OperationType NOTIFICATION and NOTIFICATION_CYCLIC.

#### 2.1.2.4.6.3.1 [VCC IP Prot: 0039/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how static notifications (using OperationType: NOTIFICATION) shall be implemented using UDP. |
| Source | |
| Verification Method | Test |

**Req:** The application shall always send at least one successful NOTIFICATION message (ACK message must have been received) after the link is established.

**Note:** It's up to each application (ServiceID.OperationID) on the subscriber side (client) to decide if there shall be a delay before the notification's acknowledge message is sent. If specific delays are needed for notification messages, this shall clearly be stated in the VCC signal specification (VCC IP Command Bus), since they affect the "wait-for-ack" timer at the server side.

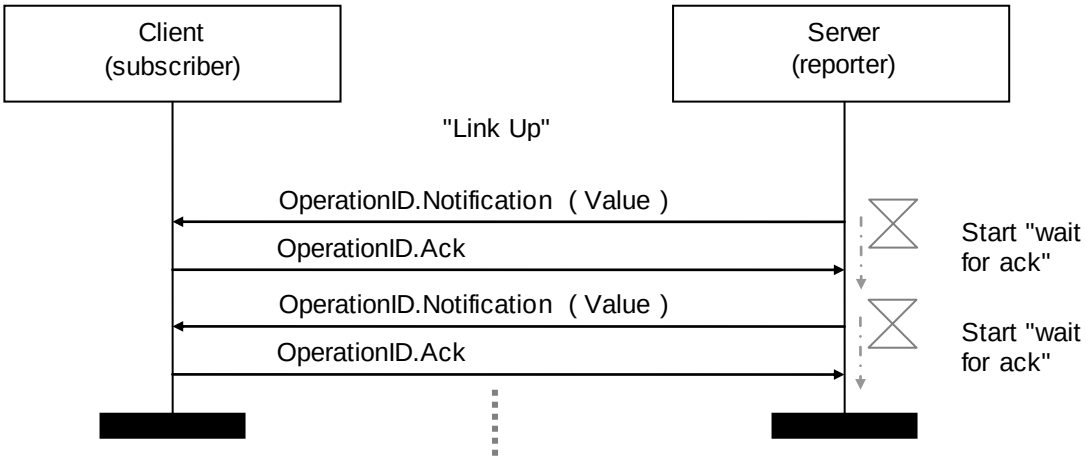



**Figure 20: Message Sequence Format – Event-based Notification using UDP**

### 2.1.2.4.6.3.2 [VCC IP Prot: 0095;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how static notifications (using OperationType: NOTIFICATION_CYCLIC) shall be implemented using UDP. |
| Source | |
| Verification Method | Test |

**Req:** The application shall always send at least one NOTIFICATION_CYCLIC message after the link is established.

**Note:** It's up to each application (ServiceID.OperationID) on the subscriber side (client) to decide the value of *T*, which specifies the transmission frequency of the notification message.

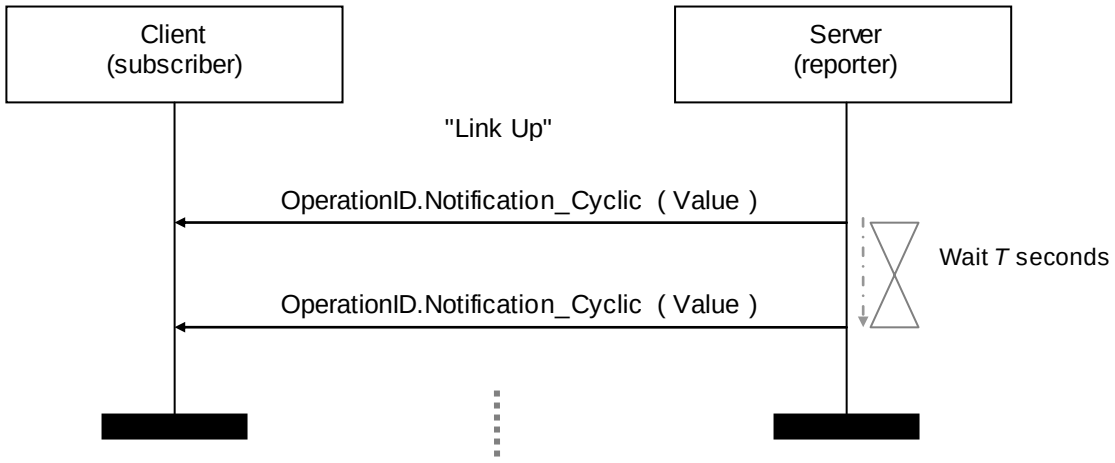



**Figure 21: Message Sequence Format – Cyclic Notification using UDP**

### 2.1.2.4.6.4 TCP

The following section explains the details when TCP is used as transport protocol for OperationType `NOTIFICATION` and `NOTIFICATION_CYCLIC`.

#### 2.1.2.4.6.4.1 [VCC IP Prot: 0040/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how static notifications (using OperationType: `NOTIFICATION`) shall be implemented using TCP. |
| Source | |
| Verification Method | Test |

**Req:** The application shall always send at least one successful `NOTIFICATION` message after the link is established.

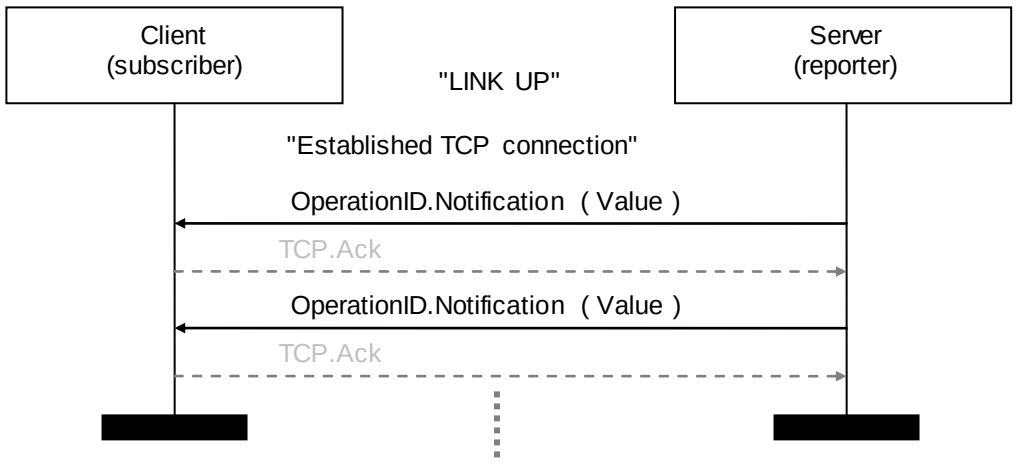



**Figure 22: Message Sequence Format – Static Notification using TCP**

### 2.1.2.4.6.4.2 [VCC IP Prot: 0096;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how static notifications (using OperationType: NOTIFICATION_CYCLIC) shall be implemented using TCP. |
| Source | |
| Verification Method | Test |

**Req:** The application shall always send at least one successful NOTIFICATION_CYCLIC message after the link is established.

**Note:** It's up to each application (ServiceID.OperationID) on the subscriber side (client) to decide the value of *T* which specifies the transmission frequency of the notification message.

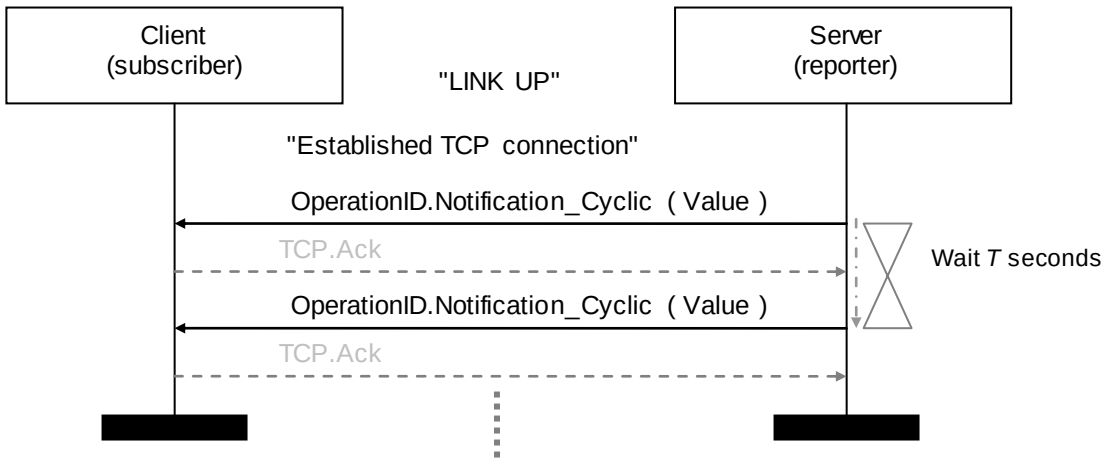



**Figure 23: Message Sequence Format – Static Cyclic Notification using TCP**

### 2.1.2.4.7 OperationType NotificationRequest

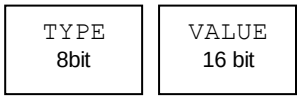
Notification request are used to support a "dynamic subscription" behavior (not hard-coded behavior) where the clients may at any time ask to subscribe on a certain operation. If a notification request is successful, the client has started to subscribe on a certain service/operation and it's expected that notification messages will be received. If connection to the serving node for any reason is cut off the subscriber needs to request to subscribe again to resume the information flow.

#### 2.1.2.4.7.1 [VCC IP Prot: 0041/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how notifications request (dynamic subscription) shall be implemented. |
| Source | [VCC IP Prot: 0038/;-1], [VCC IP Prot: 0039/;-1], [VCC IP Prot: 0095;-0], [VCC IP Prot: 0040/;-1], [VCC IP Prot: 0096;-0] |
| Verification Method | Test |

**Req:** NOTIFICATION_REQUEST messages shall be followed up with notification messages, which are either using OperationType NOTIFICATION or NOTIFICATION_CYCLIC on a specific service.

The payload of the message using OperationType NOTIFICATION_REQUEST shall use the following structure:




TYPE defines what kind of subscription action is needed on the server. Details on the accepted values can be found below in *Table 8: Presenting Type values used by Notification Request messages*

VALUE represents one OperationID of which the notification object is requested.

| TYPE | FriendlyName | Description |
|---|---|---|
| 0x00..0x04 | RESERVED | *Reserved for future usage* |
| 0x05 | NOTIFICATION | Request to activate subscription on specific OperationID(s) according to VALUE. |
| 0x06 | NOTIFICATION_CYCLIC | Request to activate subscription on specific OperationID(s) according to VALUE. |
| 0x07..0xFD | RESERVED | *Reserved for future usage* |
| 0xFE | STOP SPECIFIC SUBSCRIBTION | Stops subscription on a specific VALUE |
| 0xFF | STOP ALL SUBSCRIBTIONS | Stops all subscription on the message used SERVICEID. |

**Table 8: Presenting Type values used by Notification Request messages**

### 2.1.2.4.7.2 [VCC IP Prot: 0098/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how notifications request (dynamic subscription) shall be implemented. |
| Source | [VCC IP Prot: 0038/;-1], [VCC IP Prot: 0097/;-0] |
| Verification Method | Test |

**Req:** Applications subscribing to notifications dynamically shall consider subscription terminated if the requested service becomes *Unavailable* (according to requirement [VCC IP Prot: 0082/;-0]). This means that the application must send a new `NOTIFICATION_REQUEST` message to resume the information flow when the service becomes *Available*.

### 2.1.2.4.7.3 UDP

The following section explains the details when UDP is used as transport protocol for OperationType `NOTIFICATION_REQUEST`.

### 2.1.2.4.7.4 [VCC IP Prot: 0042/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how notifications request shall be implemented using UDP. |
| Source | [VCC IP Prot: 0041/;-1] |
| Verification Method | Test |

**Req:** When the client sends the `NOTIFICATION_REQUEST` message and request to subscribe on a certain OperationID on a server, the client expects an `ACK` message in return. When the `ACK` message has been received, the client further expects to receive a `NOTIFICATION` or `NOTIFICATION_CYCLIC` messages from the server according to the *Figure 24: Message Sequence Format – NotificationRequest using UDP.*

If the server receives a faulty/erroneous `NOTIFICATION_REQUEST` message, the server shall send an `ERROR` message instead of `ACK` and send no additional `NOTIFICATION` messages. The client needs to re-transmit the `NOTIFICATION_REQUEST`.

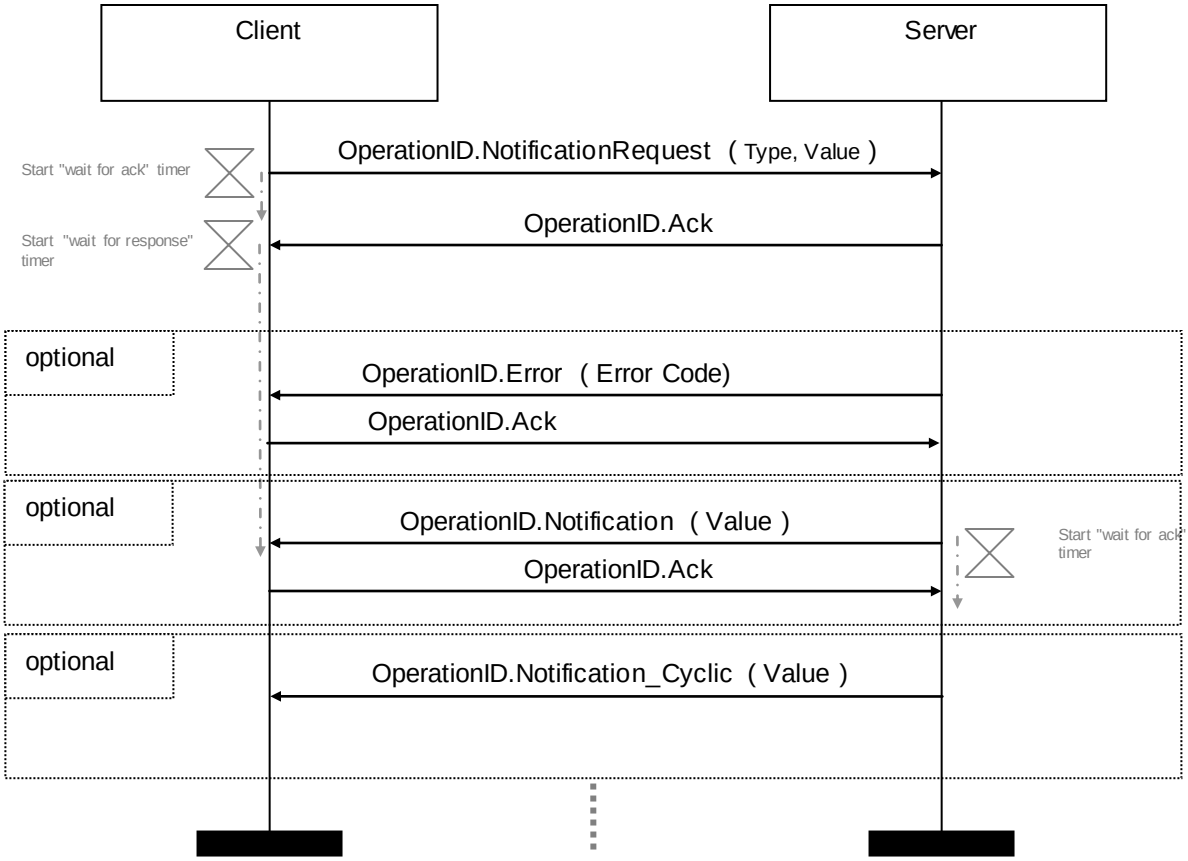



**Figure 24: Message Sequence Format – NotificationRequest using UDP**

### 2.1.2.4.7.5 TCP

The following section explains the details when TCP is used as transport protocol for OperationType NOTIFICATION_REQUEST.

### 2.1.2.4.7.6 [VCC IP Prot: 0043/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how notifications request shall be implemented using TCP. |
| Source | [VCC IP Prot: 0041/;-1] |
| Verification Method | Test |

**Req:** When the client sends the NOTIFICATION_REQUEST message and request to subscribe on a certain OperationID on a server, the client expect to receive NOTIFICATION or NOTIFICATION_CYCLIC messages from the server according to the *Figure 25: Message Sequence Format – NotificationRequest using TCP*.

If the client receives an ERROR message instead of NOTIFICATION, it needs to retransmit the NOTIFICATION_REQUEST before the subscription can be considered started.

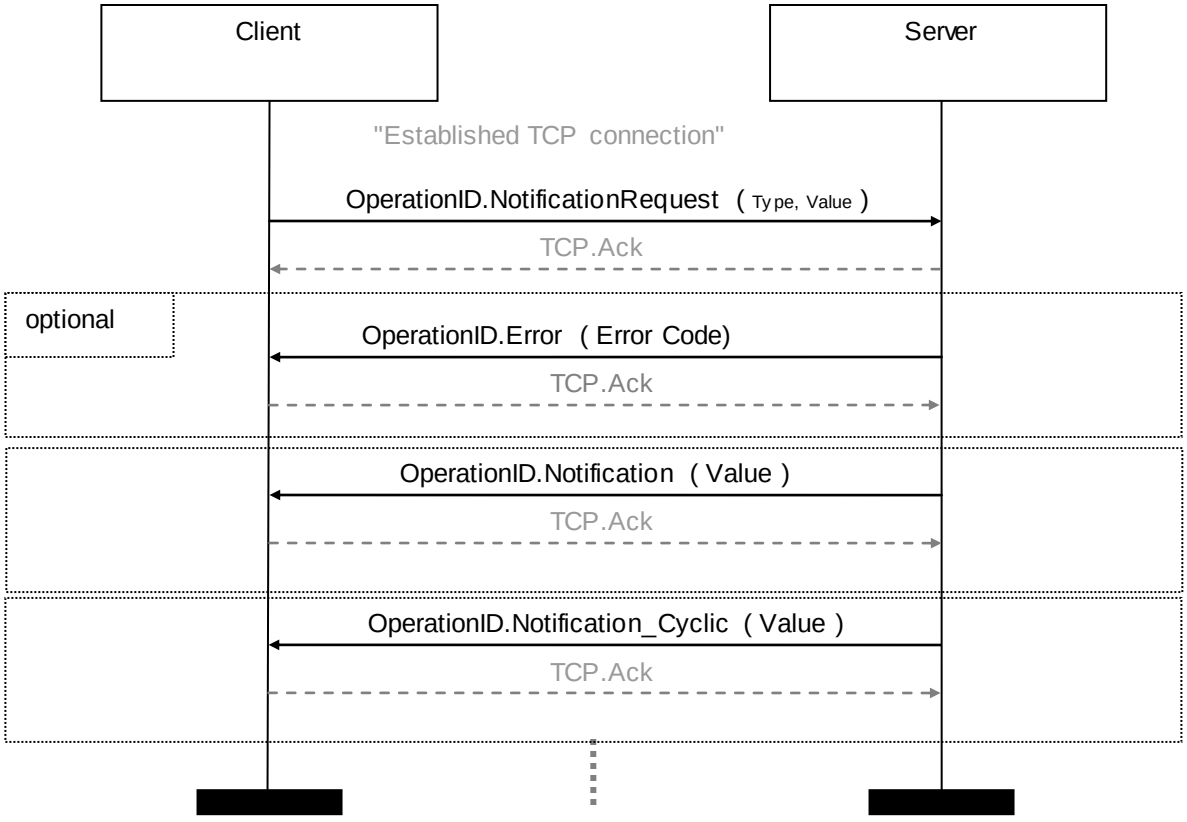



**Figure 25: Message Sequence Format – NotificationRequest using TCP**

## 2.1.3 Message Handling

### 2.1.3.1 Timers

When UDP is used as transport protocol, there is a need to be able to detect lost packets and to perform retransmission. Using timers in both the client and the server which is responsible for triggering retransmissions, UDP becomes reliable.

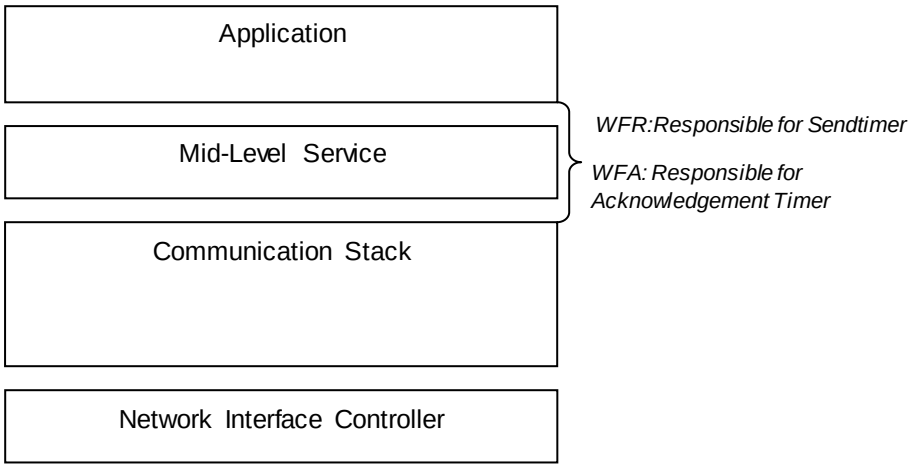



**Figure 26: Timers allocation on the OSI-model**

#### 2.1.3.1.1 [VCC IP Prot: 0044/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the acknowledgment transmission timers used for UDP transmission. |
| Source | |
| Verification Method | Test |

**Req:** Applications shall implement an acknowledgment timer (WFA), which is responsible for guaranteeing transmission of UDP messages by using `ACK` messages. If no `ACK` message is received, the application shall re-transmit the previous message.

**Note**: On the client side, there are exceptions where the `ERROR` or `RESPONSE` message shall be interpreted as an `ACK` message, in cases when these arrive prior to the ACK message itself.

### 2.1.3.1.2 [VCC IP Prot: 0045/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the response timer which shall be used on normal requests used for transmission for either UDP or TCP. |
| Source | |
| Verification Method | Test |

**Req:** For either UDP or TCP, applications shall implement a wait-for-response (WFR) timer, which is responsible for guaranteeing a certain message is responded to according to the sequence definitions (defined in this specification), within a specific time according to the chapter *Application Message Format Sequences*.

E.g.: If a `REQUEST` is performed there should be a `RESPONSE` within x milliseconds.

It's up to each application to decide the response timer behaviour. This specification will propose a "standard" behaviour which shall be implemented if nothing else is stated. More information can be found in chapter 2.1.3.1.4 Response Timer (WFR)
.

### 2.1.3.1.3 Acknowledgement Timer (WFA)

### 2.1.3.1.3.1 UDP

The "Wait-For-Acknowledgement" (WFA) timer is used whenever a sender sends a message as UDP packets. The WFA timer shall be used to regulate the amount of time that the application shall wait on an acknowledgment message before resending (more info about retransmission can be found in chapter 2.1.3.1.4.4).

The `SenderHandleID` in combination with the source IP-address is a good way to identify which messages that expects an acknowledgment message. The WFA shall be a lower level retry mechanism which is handled by the layers below the application.

#### 2.1.3.1.3.1.1 [VCC IP Prot: 0046/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the applications shall handle `ACK` messages, when UDP is used as transport protocol. |
| Source | |
| Verification Method | Test |

**Req:** The receiving application shall respond with an `ACK` message (after the health check) before processing the message.

- On the sender side, the application shall set a WFA-timer before sending a message.
  The WFA timer shall be terminated upon an `ACK` message matching the initiating message. Note: in some cases, `ERROR` or `RESPONSE` messages are interpreted as `ACK` messages.

- On the receiver side, upon reception of a new message, a health check is performed to detect errors and after that, the `ACK` message shall immediately be sent out before processing the message.

#### 2.1.3.1.3.1.2 [VCC IP Prot: 0047/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the default acknowledgement timeout values, when UDP is used as transport protocol. |
| Source | |
| Verification Method | Test |

**Req:** The default acknowledgment timeout value shall be named, `defaultTimeoutWFA`.
This timer shall be set to 500 (TBD) milliseconds as default value.

**Note:** The `defaultTimeoutWFA` value shall be adjustable via a local configuration file within each ECU that uses the IP command bus.

More information about retransmission formula is stated in chapter 2.1.3.3.2.1

### 2.1.3.1.3.1.3 [VCC IP Prot: 0048/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how to behave upon application restart or transmission success, when UDP is used as transport protocol. |
| Source | |
| Verification Method | Test |

**Req:** If the application is restarted or the transmission has been successful (ACK message received), the default acknowledgment timeout value (defaultTimeoutWFA) shall be used.

### 2.1.3.1.3.1.4 [VCC IP Prot: 0049/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how to retransmit messages when the acknowledgment timer has expired, when UDP is used as transport protocol. |
| Source | |
| Verification Method | Test |

**Req:** An application which sends a control message shall create the message and put it to the transmit buffer. The application shall also start the WFA timer which counts down until an ACK message is received or the timer value has expired.

If the sender application does not receive an ACK message within a certain time, an ACK timeout occurs. The application shall then retransmit the control message.

**Note:** There might be scenarios where the server is not available due to a reset. It's therefore important to use the acknowledgment mechanism (WFA) to make sure that the message is delivered.

The *Figure 27: Message Sequence Format – OperationType Request with "ACK" time-out*, is based on an example where the client sends a REQUEST message. The example shows a sequence when a WFA timer times out.
**Note:** The WFR timer is not include in this example but covered in the example in chapter 2.1.3.1.4.

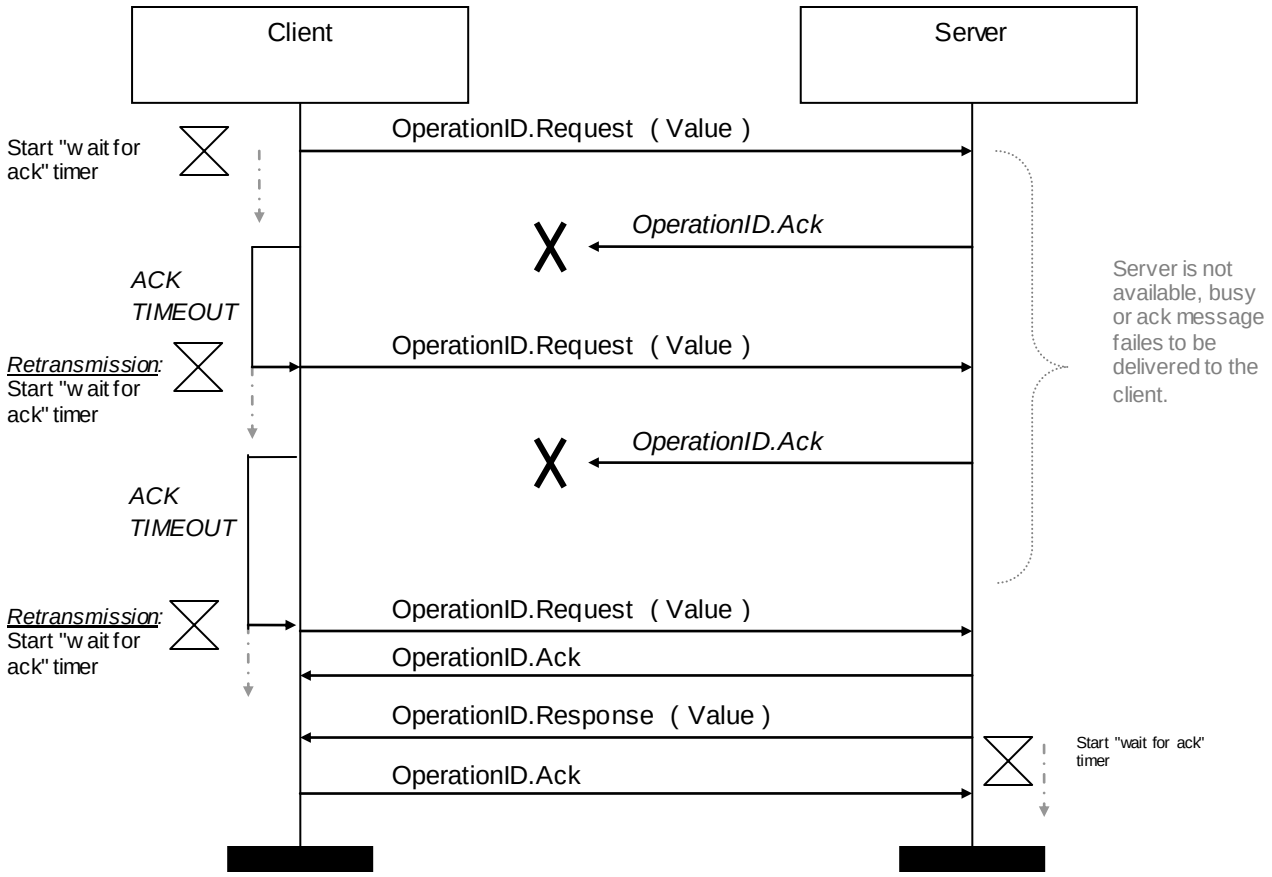



**Figure 27: Message Sequence Format – OperationType Request with "ACK" time-out**

#### 2.1.3.1.4 Response Timer (WFR)

The "wait-for-response" timer (WFR) is used if an application uses a control message which requires a response within a certain time, independently if UDP or TCP is used as transport protocol.

The WFR implementation is application specific and must suit a specific use-case.

##### 2.1.3.1.4.1 [VCC IP Prot: 0050/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the default response timeout value (WFR), using either UDP or TCP. |
| Source | |
| Verification Method | Test |

**Req:** The default response timeout value shall be named defaultTimeoutWFR.
This timer shall be set to 500 (TBD) milliseconds as default value.

**Note:** The defaultTimeoutWFR value shall be adjustable via a local configuration file within each ECU that uses the IP command bus.

More information about retransmission formula is stated in chapter 2.1.3.3.2.1

##### 2.1.3.1.4.2 [VCC IP Prot: 0051/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how a retransmission shall be implemented using the WFR timers, when either UDP or TCP is used. |
| Source | |
| Verification Method | Test |

**Req:** If the application is restarted or the transmission has been a success (response-received), the default response timeout value (defaultTimeoutWFA) shall be used.

#### 2.1.3.1.4.3 UDP

##### 2.1.3.1.4.3.1 [VCC IP Prot: 052/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the applications shall handle the WFR timer, when UDP is used as transport protocol. |
| Source | |
| Verification Method | Test |

**Req:** When UDP is used, the WFR timers shall be implemented as the following:

On the client side, when a REQUEST (or SETREQUEST) message have been sent, the application shall after receiving the ACK message start a WFR-timer.

If a RESPONSE message is received, the WFR-timer shall be terminated.

### 2.1.3.1.4.3.2 [VCC IP Prot: 0053/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how a retransmission shall be implemented using the WFR timers, when UDP is used. |
| Source | |
| Verification Method | Test |

**Req:** If WFR-timeout occurs, the WFA timer (defaultTimeoutWFA) shall be reset to default value. Before the retransmission, the default wait-for-acknowledgment timer (defaultTimeoutWFA) shall be reset to default value. More information about retransmission can be found in chapter 2.1.3.1.3.

### 2.1.3.1.4.3.3 Example – Wait-for-Response using UDP

The following sequence shows an example when a WFR (Wait-For-Response) timer times out.
The client sends a `REQUEST` message and receives an `ACK` message from the server. The client starts the WFR-timer and waits for a `RESPONSE` message. If the client does not receive the `RESPONSE` message within the appropriate time, it shall set the WFA timer to `defaultTimeoutWFA` and then retransmit the `REQUEST`.

The number of necessary retries is defined in chapter 2.1.3.3.
**Note:** The "WFA-timers" are greyed by intention in the example below.

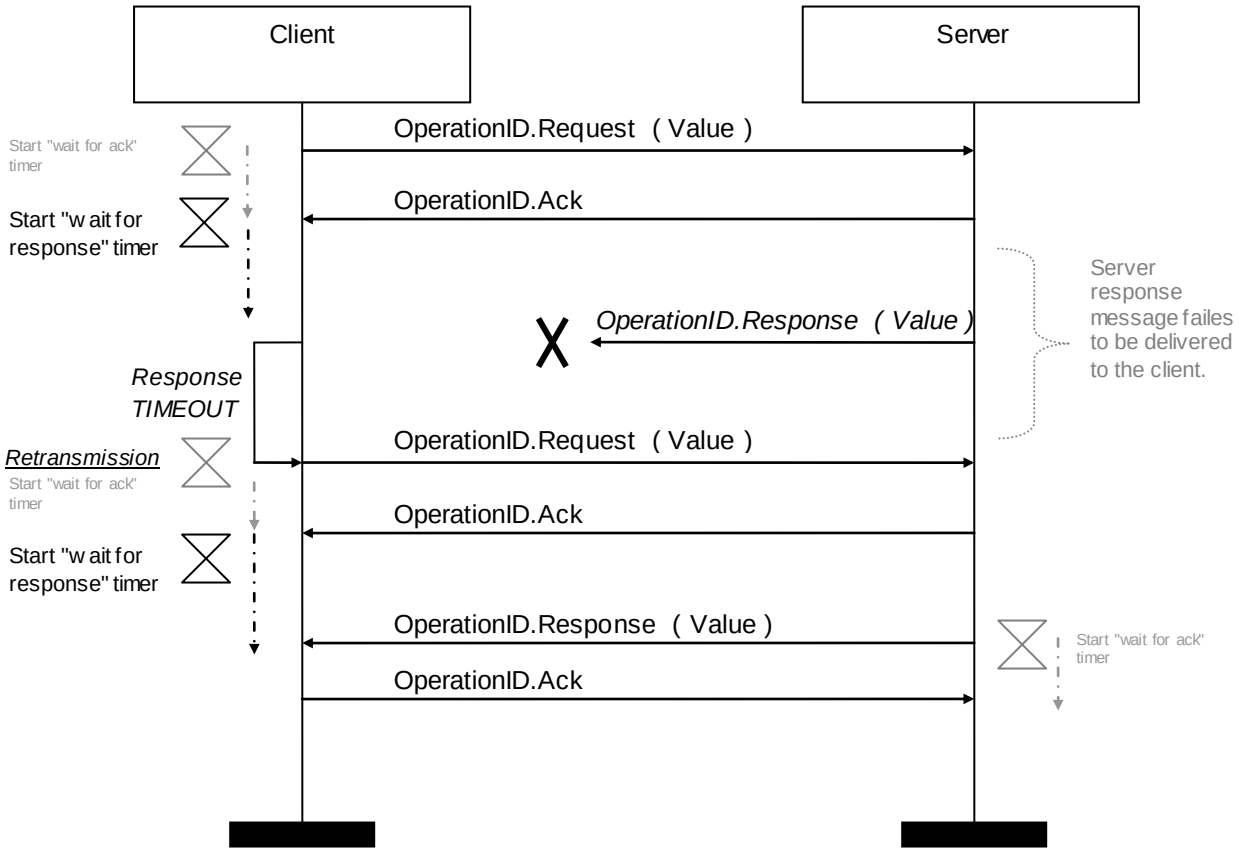



**Figure 28: Message Sequence Format – OperationType Request with WFR time-out using UDP**

## 2.1.3.1.4.4 TCP

### 2.1.3.1.4.4.1 [VCC IP Prot: 054/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the applications shall handle the WFR timer, when TCP is used as transport protocol. |
| Source | |
| Verification Method | Test |

**Req:** When TCP is used, the WFR timers shall be implemented as the following:

On the client side, before sending a `REQUEST` (or `SETREQUEST`) message, the application shall start the WFR-timer. When the `RESPONSE` message is received, the WFR-timer shall be terminated. If the timer times out the `REQUEST` message shall be retransmitted after the timer is reset.

*Figure 29* presents an example on how the WFR timers shall be used.

### 2.1.3.1.4.4.2 Example – Wait-for-Response using TCP

The following sequence shows an example when a WFR (Wait-For-Response) times out when TCP is used. The client starts the WFR-timer and sends a `REQUEST` message and waits for a `RESPONSE` message. If the client does not receive the appropriate message (`RESPONSE`) within the appropriate time, it shall retransmit the message.

The number of necessary retries is defined in chapter 2.1.3.3.

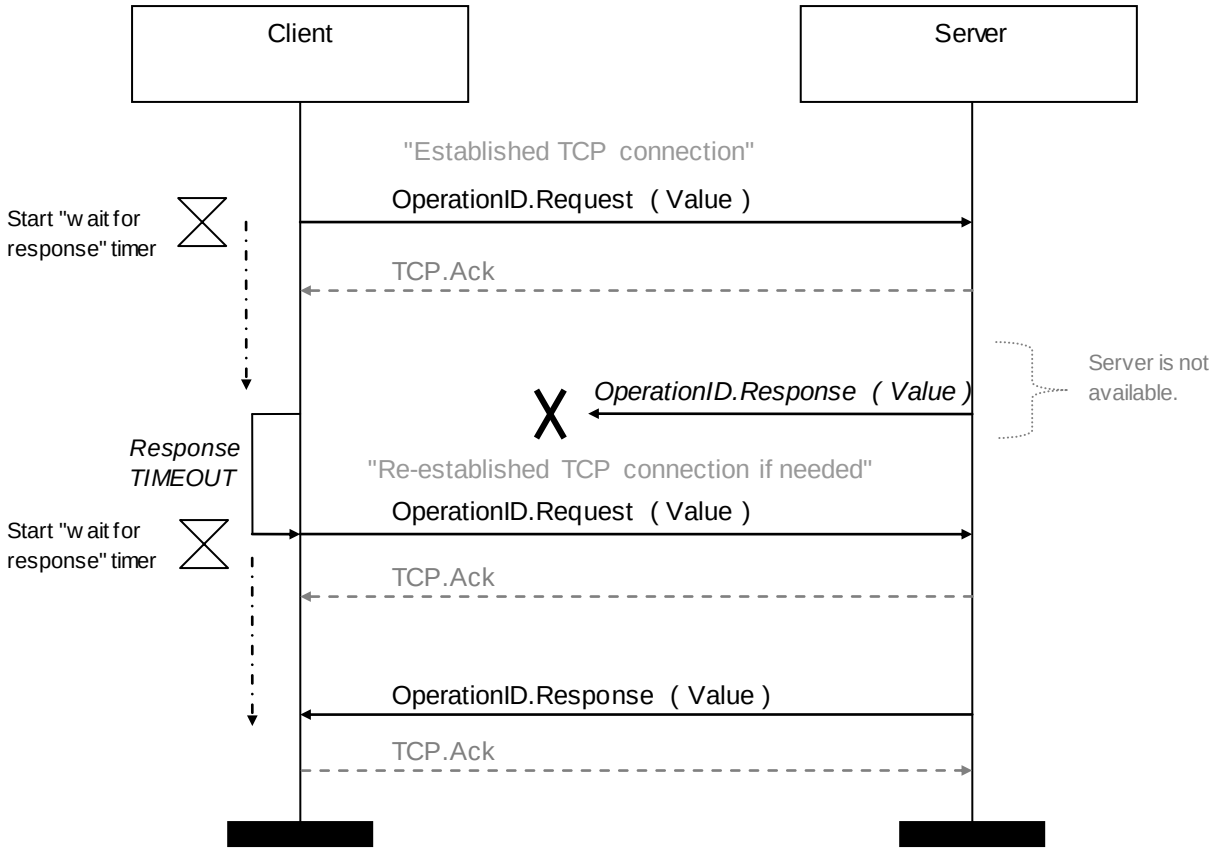



**Figure 29: Message Sequence Format – OperationType Request with WFR time-out using TCP**

## 2.1.3.2 Concurrent Message handling

### 2.1.3.2.1 [VCC IP Prot: 0055/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the application behaviour with respect to simultaneously/concurrent message sequences. |
| Source | |
| Verification Method | Test |

**Req:** An application shall be able to handle multiple incoming and outgoing messages in parallel (concurrently).

This means that if a application sends a message with OperationType: `REQUEST`, the application shall not only be able to wait for the `ACK`, `RESPONSE` or `ERROR` message, but handle multiple incoming/outgoing messages at the same time.

### 2.1.3.2.2 [VCC IP Prot: 0056/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the minimum concurrent sequences |
| Source | |
| Verification Method | Test |

**Req:** Each service shall be able to handle at minimum 10 (TBD) simultaneously operation's (concurrent message sequences) at the same time.

### 2.1.3.2.3 [VCC IP Prot: 0057/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the behaviour if all slots are busy |
| Source | |
| Verification Method | Test |

**Req:**
If a message (e.g. `REQUEST`) is sent addressing an "active" OperationID (application is currently processing a sequence), or if all "slots" are busy on the application, the receiver application shall respond with an `ERROR` message using the `ErrorCode`: "busy" see table in chapter 2.1.4.

Example:
A Service receives a OperationType "SetRequest" (Operation ID: 0xA0) and is handling/processing that request. Meanwhile a second SetRequest is received with the exact same OperationID: 0xA0. The application can return an Error message informing "I'm busy".

### 2.1.3.3 Retransmission of messages

If the network is congested and many messages are dropped, retransmissions might create even more load into the system (so more messages will be dropped or delayed). If a sender fails with the retries of the WFA (numberOfRetriesWFA) and/or WFR (numberOfRetriesWFR), it's important to relax the network of retransmissions.

#### 2.1.3.3.1 [VCC IP Prot: 0058/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the WFA-retransmissions when UDP is used. |
| Source | |
| Verification Method | Test |

**Req:** If UDP is used, there shall be a retry counters which counts the maximum number of WFA-retransmissions. The retry counter shall be set to:

- numberOfRetriesWFA = 7

More information about retransmission formula is stated in chapter 2.1.3.3.2.1

#### 2.1.3.3.2 [VCC IP Prot: 0059/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the WFR-retransmissions used by UDP and TCP. |
| Source | |
| Verification Method | Test |

**Req:** There shall be a retry counters which counts the maximum number of WFR-retransmissions. The retry counter shall be set to:

- numberOfRetriesWFR = 2

More information about retransmission formula is stated in chapter 2.1.3.3.2.1

### 2.1.3.3.2.1 WFA and WFR - Retransmissions formula

The following chapter defines the retransmission formula which shall be used for WFA or WFR.

#### 2.1.3.3.2.1.1 [VCC IP Prot: 0060/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the retransmission formula which shall be used for WFA and WFR |
| Source | |
| Verification Method | Test |

**Req:** The following retransmission formula shall be used.

usedTimeoutWFx = defaultTimeoutWFx * ( increaseTimerValueWFx ^ numberOfRetriesWFx )

Where:

- WFx is either WFA or WFR as applicable
- usedTimeout is the time the sender will wait for ACK/reply on this transmission
- numberOfRetries is 0 for the first transmission and increased by 1 for each retry.

increaseTimerValueWFA = 1,5
increaseTimerValueWFR = 2

- If the client has performed maximum allowed retries and failed due to WFA timeouts, both the WFA timer/counter and the WFR timer/counter shall be restored to default values.
- Upon a reset of the devices or reception of appropriate acknowledgment (ACK) or response (RESPONSE), the application shall load the default defaultTimeoutWFA and defaultTimeoutWFR timer values.

  usedTimeoutWFx = defaultTimeoutWFx

It's then up to the application to decide if it should restart the transmission trials or not.

**Note:** The defaultTimeoutWFA, defaultTimeoutWFR, numberOfRetriesWFA, numberOfRetriesWFR, increaseTimerValueWFA and increaseTimerValueWFA shall be adjustable via a local configuration file within each ECU that uses the IP command bus.
In addition, is shall be possible to specify the defaultTimeoutWFR and numberOfRetriesWFR values uniquely per VCC IP Command Bus signal, in which the default values are replaced by values specified in the corresponding SWRS and SRD only for the specific signal.

### 2.1.3.3.2.2 Example of retransmission times using WFA or WFR.

The following example used the following default WFA and WFR values:

defaultTimeoutWFA: 500ms
increaseTimerValueWFA: 1,5
numberOfRetriesWFA = 6
defaultTimeoutWFR: 500ms
increaseTimerValueWFR: 2
numberOfRetriesWFR = 2

The table below shows that after the 6th retransmission using WFA, the total waiting time should be ~ 16 seconds (WFA-Accumulated time).

| | usedTimeoutWFA (ms) | WFA - Accumulated time (ms) | usedTimeoutWFR (ms) | WFR -Accumulated time (ms) |
|---|---|---|---|---|
| First transmission ( 0 retries) | 500 | 500 | 500 | 500 |
| Second transmission (1 retry) | 750 | 1250 | 1000 | 1500 |
| Third transmission (2nd retry) | 1125 | 2375 | 2000 | 3000 |
| Fourth transmission (3rd retry) | 1687,5 | 4062,5 | - | - |
| Fifth transmission (4th retry) | 2531,25 | 6593,75 | - | - |
| Sixth transmission (5th retry) | 3796,875 | 10390,63 | - | - |
| Seventh transmission (6th retry) | 5695,313 | 16085,94 | - | - |
| Eight transmission (7th retry) | 8542,969 | 24628,9 | | |

**Table 9: Retransmission showing usedTimeoutWFx and the accumulated waiting time for WFx**

### 2.1.3.3.3 Message Handling

The current subchapter shall be seen as a good practice of how to design the message handling on the applications. It's not a requirement but a guideline.

#### 2.1.3.3.3.1 Client

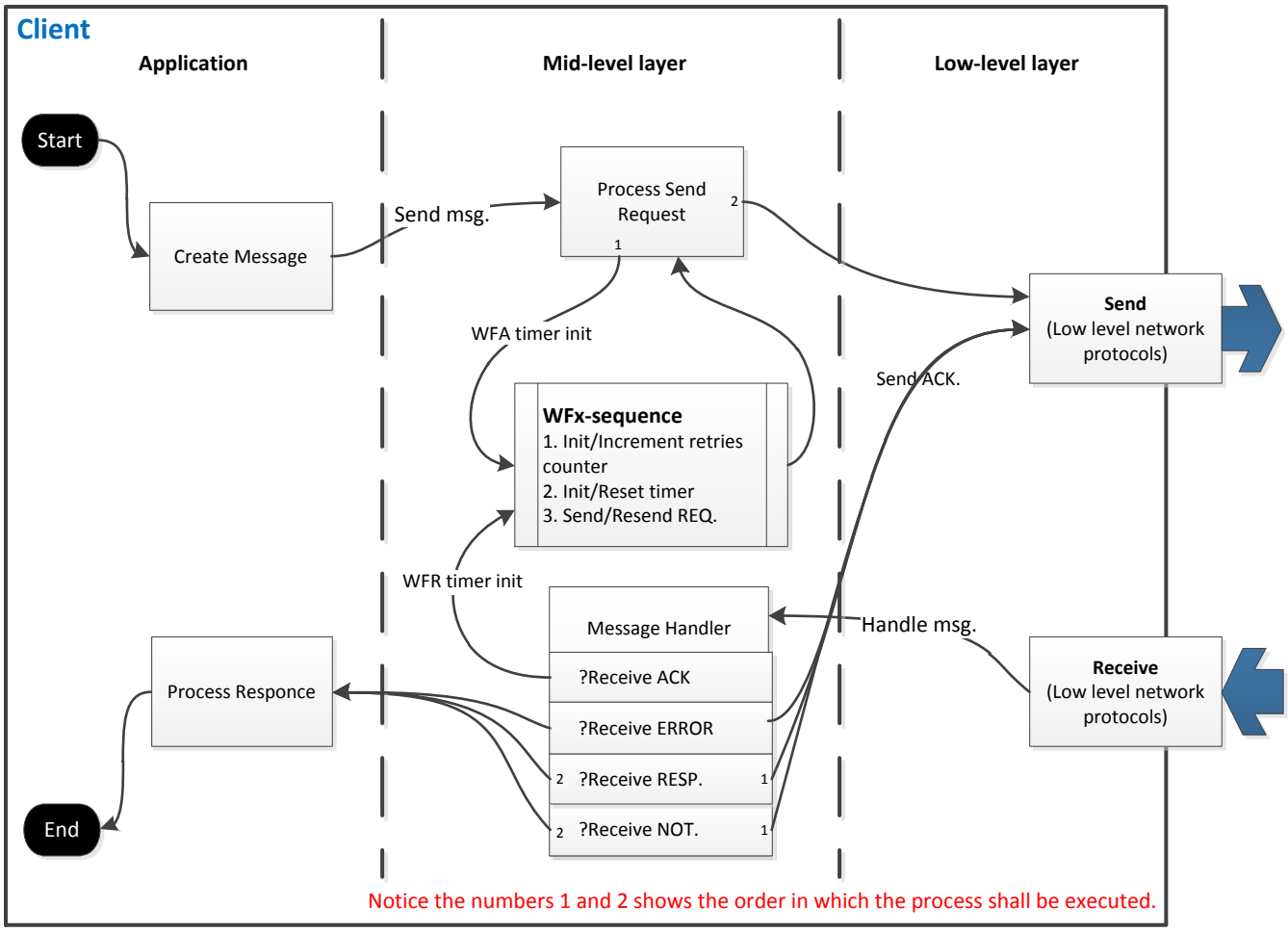



**Figure 30: Presenting a client flowchart**

More information about the message processing mechanism can be found in chapter: 2.1.3.3.3.3.

### 2.1.3.3.3.2 Server

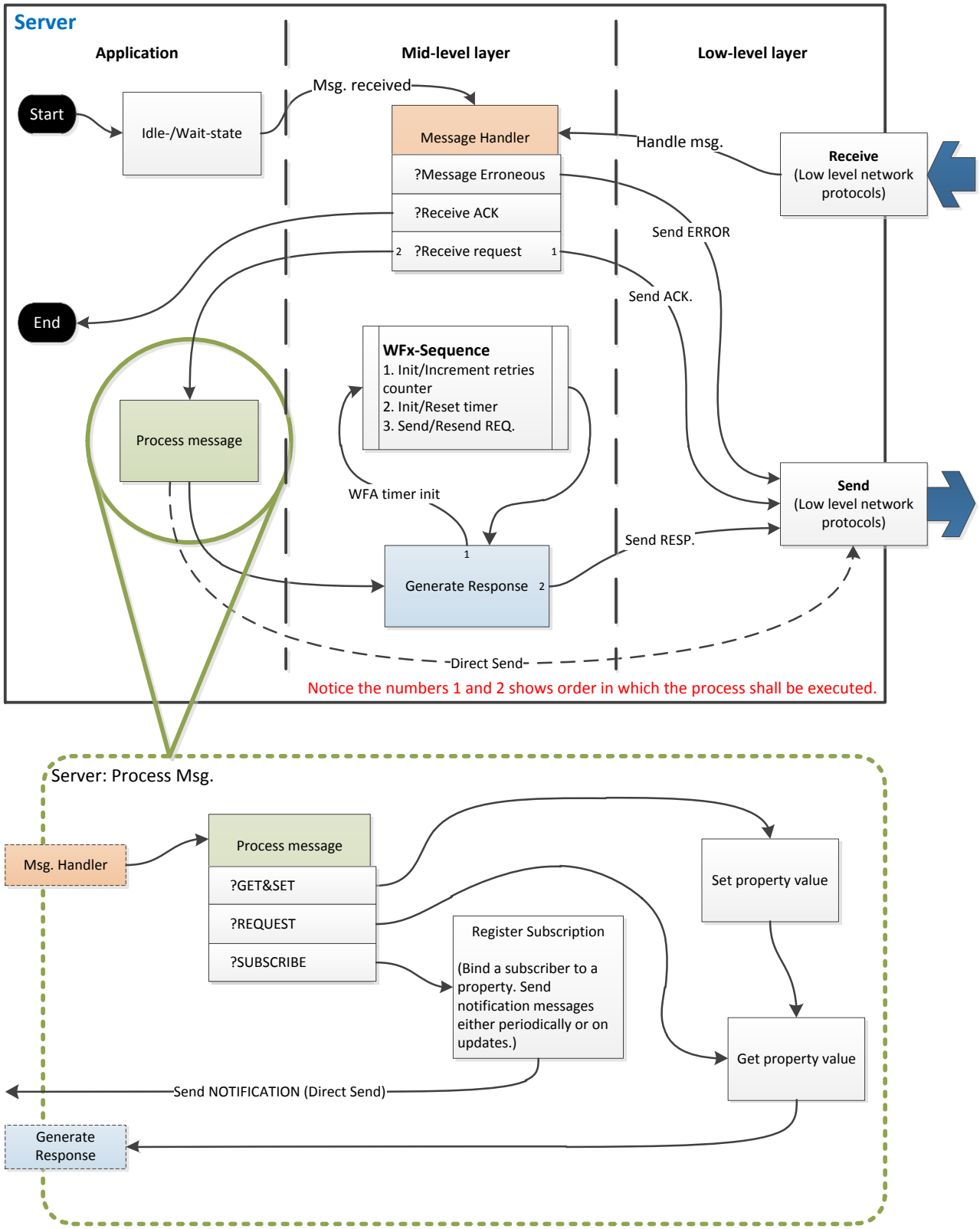



**Figure 31: Presenting a server flowchart**

More information about the message processing mechanism can be found in chapter: 0

### 2.1.3.3.3.3 Client side – Message processing mechanism

## Application Layer

#### 2.1.3.3.3.3.1 Start

The client sends an application message of some type. The possible requests for the client are stated in chapter 2.1.2.3.7 and Table 6: Presenting valid Operation Types.

#### 2.1.3.3.3.3.2 Create message

The message is created and the client forwards it to the *midlevel* layer by invoking the application level send function. The midlevel layer is responsible for keeping track of sent messages by managing the WFA and WFR timers that tells about missing responses from the server.

#### 2.1.3.3.3.3.3 Process response

At this state the application shall process the response according to application requirements and then end the message handling procedure.

#### 2.1.3.3.3.3.4 End

There shall not be any such state in the implementation. Its purpose is to visualize the end of this scope.

## Mid-Level Layer

#### 2.1.3.3.3.3.5 Process Send Request

This mechanism shall be implemented in the midlevel layer. Being in this state means that the message is about to be sent to the server. The client application shall enter the WFA-sequence that is described below, before the message is forwarded.

#### 2.1.3.3.3.3.6 Message handler

**Note,** the question marks in front of the message type in the flow chart are to be read as "If" and are to clarify the different ways a message may be handled.

Being in this state is an indication that a message has arrived and needs to be handled. The client is expected to handle five types of messages in the standard implementation, namely the `ACK` (Acknowledgement), `RESPONSE`, `NOTIFICATION`, `NOTIFICATION_CYCLIC` and the `ERROR` messages. First of all, the message handler shall check the message if there are any errors.

If a `RESPONSE`, `ERROR` or `NOTIFICATION` message is received, the application shall reply with an `ACK` message before continuing to the *Process response* state described below.

`ACK` messages are received as result of successfully transmitted messages. In case the message is expected to be replied the application shall enter the *WFR-sequence* where it shall initiate "wait for response" sequence.

#### 2.1.3.3.3.3.7 WFx-sequence

WFx-sequence is explained in chapter: 2.1.3.3.3.5

## Low-Level Layer

### 2.1.3.3.3.8 Send

This state is common for both the client and the server and defines the mechanism that does the necessary steps to forward the message through the stack to the destination IP ECU. This state can be ignored when implementing the application. Its purpose is to clarify the path taken by the message.

### 2.1.3.3.3.9 Receive

Low level message handling.

## 2.1.3.3.3.4 Server side – Message processing mechanism

## Application Layer

### 2.1.3.3.3.4.1 Start

This is included only to visualize a starting point of the procedure.

### 2.1.3.3.3.4.2 Idle-/wait-state

At this state the server application is idling or waiting for an incoming message to handle. This might even be a separate receiver thread in the application that is woken by the network stack when a message is received leading to the message handler.

### 2.1.3.3.3.4.3 Send

This send state is the same as for the client.

### 2.1.3.3.3.4.4 End

End indicated the end of a session which technically means for the server to enter the *Idle-/wait*-state to wait for a new message to arrive.

### 2.1.3.3.3.4.5 Process message

This state is part of the message handler procedure in the sense that it distinguishes the different types of request messages.
Messages of operation type `REQUEST` shall be forwarded to the *Get Property state*.
`SETREQUEST_NORETURN` and `SETREQUEST` messages shall be forwarded to the *Set Property* state.
`NOTIFICATIONREQUEST` messages shall be forwarded to the *Register subscription* state.

#### 2.1.3.3.3.4.5.1 Set property value

Messages requesting to set new values for specific properties reach this state. There are two outcomes from this state; the client is either only requesting to set a new value in which case it shall be done and then continued by ending the session; or the client is requesting to set a new value and asking for a response containing the value just set to the given property in which case it shall set the new value and then continue to the *Get property* state.

#### 2.1.3.3.3.4.5.2 Get property value

Any message getting to this state is requesting the value of a specific property. Here the application shall get the requested property value and proceed with the *Generate response* state.

#### 2.1.3.3.3.4.5.3 Register subscription

Client applications may be interested of a property and especially when its value is updated, in which case they send subscription requests. These requests are processed at this state and the implementation involving these subscriptions are more complex than explained here.
As stated by the requirement on the subscription mechanism in chapter 2.1.2.4.5, the value shall be read either periodically or on events triggered by updates depending on which type is chosen. A notification message of the chosen type shall be created that include this value which is directly sent to the client via the *Send* state.

## Mid-Level Layer

### 2.1.3.3.3.4.6 Generate response

In this state the application shall create the response message and then enter the WFA-sequence. This response message is then sent to the client via the *Send* state. See the *WFx-sequence* to understand the possible scenarios.

### 2.1.3.3.3.4.7 Message handler

**Note,** the question marks in front of the message type in the flow chart are to be read as "If" and are to clarify the different ways a message may be handled.

The server application's message handler can handle messages of type `REQUEST`, `SETREQUEST_NORETURN`, `SETREQUEST`, `NOTIFICATIONREQUEST` and `ACK` messages and each of these message types require different handling. Detailed requirements for the different messages are stated in section, 2.1.2.3.7 and Table 6: Presenting valid Operation Types.

As stated in requirement chapter 2.1.2.4.1.2 the message shall be sanity checked to make sure the message is in the correct format. If it does not pass the sanity check, an `ERROR` message shall be sent to the client via the direct path to the *Send* state.
If the server has received an `ACK` message it means that the earlier sent message was successfully transmitted and we shall end this session, visualized by continuing to the *End* state.
In the other cases we shall proceed to the *Process message* state described below.

### 2.1.3.3.3.4.8 WFx-sequence

WFx-sequence is explained in chapter: 2.1.3.3.3.5

## Low-Level Layer

### 2.1.3.3.3.4.9 Send

This state is common for both the client and the server and defines the mechanism that does the necessary steps to forward the message through the stack to the destination IP ECU. This state can be ignored when implementing the application. Its purpose is to clarify the path taken by the message.

### 2.1.3.3.3.4.10 Receive

Low level message handling.

## 2.1.3.3.3.5 Client/Server side – WFx-sequence

### 2.1.3.3.3.5.1 WFx-sequence

The following sequence of steps is common for both the client and the server. It is necessary to keep track of sent messages and handle possible problems that might occur on their respective paths. The WFx is either WFA or WFR depending on the entry state. They shall define their specific timer and counter which follow different rules. More about the requirement for the timers and counters can be studied in their respective paragraph 2.1.3.1 *and 2.1.3.2.*

Each step in the sequence has two possible ways denoted A and B.

A means that the message is to be sent the first time and B means the message need to be resent.

The following are the steps in the intended progress order.

1. A. Initiate a counter by setting it to 0 and associate it to the message.
   B. Check if the number of allowed retries is reached; if so reset the counter and report an error to the application otherwise increment the counter and proceed.

2. A. Initiate and start a timer based on a default timeout value and associate it to the message. The default timeout value is stated in the configuration file.
   B. Recalculate the timeout value depending on number of failed attempts, then reinitiate and start the timer.

3. A.& B. None of the previous steps has led to any unexpected error so invoke the send mechanism. For timer and retransmission specific requirements read the chapter 2.1.3.

## 2.1.4 General Error Codes

The table below shows the structure of the payload content of an `ERROR` (0xE0) message.
More information about good practice on how to interpret *Table 10: List of general Error Codes* can be found in *2.1.4 ASN.1 - Good practice of Error Code handling.*

***Example***: If a message contains wrong ServiceID and OperationID, the ErrorCode 0x01 has a higher ranking than Error Code 0x02. An error message containing Error Code 0x01 shall be used in this example.

***Example:*** If a client sends a message containing errors e.g. wrong length, wrong protocol version to a server.

### 2.1.4.1.1.1 [VCC IP Prot: 0060/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the structure of an error message |
| Source | |
| Verification Method | Test |

**Req:** An `ERROR` message shall always use the VCC IP Command Protocol header including `ServiceID`, `OperationID` , `SenderHandleID` and the `Length`.

### 2.1.4.1.1.2 [VCC IP Prot: 0061/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the structure of a error message |
| Source | |
| Verification Method | Test |

**Req:** An `ERROR` message payload shall be designed as the following:

| ErrorCode | ErrorInformation |
|---|---|
| [ 8 bits ] | [16 bits] |

### 2.1.4.1.1.3 [VCC IP Prot: 0062/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the structure of an error message with ErrorCodes and ErrorInformation. |
| Source | |
| Verification Method | Test |

**Req:** An `ERROR` message shall use the following ErrorCodes and ErrorInformation.

| ErrorCode [ 8 bit] | ErrorCode Description | ErrorInformation [ 16 bit] | Description |
|---|---|---|---|
| 0x00 | Not Ok | - | An unspecified error occurred |
| 0x01 | ServiceID not available | Return the ServiceID | The Service is not available or does not exist |
| 0x02 | OperationID not available | Return the OperationID | The Operation is not available or does not exist |
| 0x03 | OperationType is not available | Return the OperationType | Invalid OperationType |
| 0x04 | Invalid protocol version | Return the supported Protocol Version | Invalid protocol version, return the supported Protocol Version |
| 0x05 | Segmentation Error | Return which segments that is missing. | Segments is missing |
| 0x06 | Invalid Length | - | The length of the message is not correct |
| 0x07 | Application Error | - | Application not ready |
| 0x08 | Timeout | - | Timeout occurred |
| 0x09 | Busy | - | The server is busy and cannot process the message at the moment. Retry after time according to [VCC IP Prot: 0060/;-1]. |
| 0x0A – 0x1f | RESERVED | | Reserved for generic errors. |
| 0x20 – 0x3f | RESERVED | Specific Errorinformation for OperationIDs. | Reserved for specific errors of services and operations.<br><br>**Note**: These errors are specified in another section which defines the services and operations into detail. |

**Table 10: List of general Error Codes**

#### 2.1.4.1.2 ASN.1 - Good practice of Error Code handling

The following examples show how the error codes shall be handled for certain scenarios:

##### 2.1.4.1.2.1 Correct example message

UDP Payload: "00A10104000000090000000100020000<u>50</u>"
VCC PDU Length: 9 bytes
VCC PDU Data: 1 byte ("50")

**Expected action:** No error message shall be send.

##### 2.1.4.1.2.2 Incorrect Header

UDP Payload: "00A10104000000140000000100020000"
VCC PDU Length: 20 bytes
VCC PDU Data: 0 bytes

**Expected action:** The following Error Code shall be used; Invalid Length (0x06), due to an incomplete header or data is missing.

##### 2.1.4.1.2.3 Incorrect length

UDP Payload: "00A10104000000040000000100020000"
VCC PDU Length: 4 bytes
VCC PDU Data: 0 bytes

**Expected action:** The following Error Code shall be used; Invalid Length (0x06), due to an incomplete header.

##### 2.1.4.1.2.4 No senderHandleID included

UDP Payload: "00A10104000000000000000100020000"
VCC PDU Length: 0 bytes (no data, no part of header)
VCC PDU Data: 0 bytes

**Expected action:** Ignored. The senderHandleID is included but length says otherwise so the senderHandle is discarded. Therefore no error message can be sent.

##### 2.1.4.1.2.5 ASN.1 Decode Error

UDP Payload: "00A10104000000080000000100020000<u>50</u>01020304"
VCC PDU Length: 8 bytes (no data)
VCC PDU Data: 5 bytes (1st is correct "50")

**Expected action:** The following Error Code shall be used; ASN.1 Decode error, since all data after the header is thrown away and the decoder will work on zero length data

##### 2.1.4.1.2.6 ASN.1 Junk data

UDP Payload: "00A10104000000090000000100020000<u>50</u>01020304"
VCC PDU Length: 9 bytes
VCC PDU Data: 5 bytes (1st byte correct "50")

**Expected action:** No error message. Additional data (01020304) shall be discarded.

# 3. VCC IP LINK MANAGER

## 3.1 Introduction

The VCC IP Link Manager, LM module implements a mechanism with the purpose to inform Local Software Components, LSC (within an ECU) or software components running on other ECU's that services are available.

The basic principle is that one LM module instance will be running on each vehicle internal IP ECU and that these distributed LM instances shall be communicating with each other using VCC IP Command Protocol, while LSCs only communicate with their ECU local LM instance as shown in Figure 32.

When the ECU is initialized, the local LM module starts to periodically broadcast that all **services** that are configured to start on the ECU is available to other LM modules.
When the LM module is initialized it will discover possible existence of external LM modules that also are broadcasting their existence throughout the vehicle internal IP network. In that way, each LM module will know availability of all other LM modules.

An LSC may request a service which is handled by the local LM module which starts to periodically broadcast that a service is requested to the external LM modules. There is no mechanism for specifying which service that is requested, so the broadcast will mean that any/all services for the requested resource group are requested. If the local LM module realizes that not all expected LM instances are available, the local LM instance tries to enable them by sending a "vehicle bus ECU wakeup" system signal on the vehicle buses. The exact name of the "vehicle bus ECU wakeup" system signal is ECU dependent and therefore not described in this document. These are described in the SRD for "Infotainment Platform".
LSCs may also request services with high priority, in which case the local LM instance adds this information to its broadcasted Request messages. The external LM modules then realize that the received Request has high priority thus should ignore normal power handling.

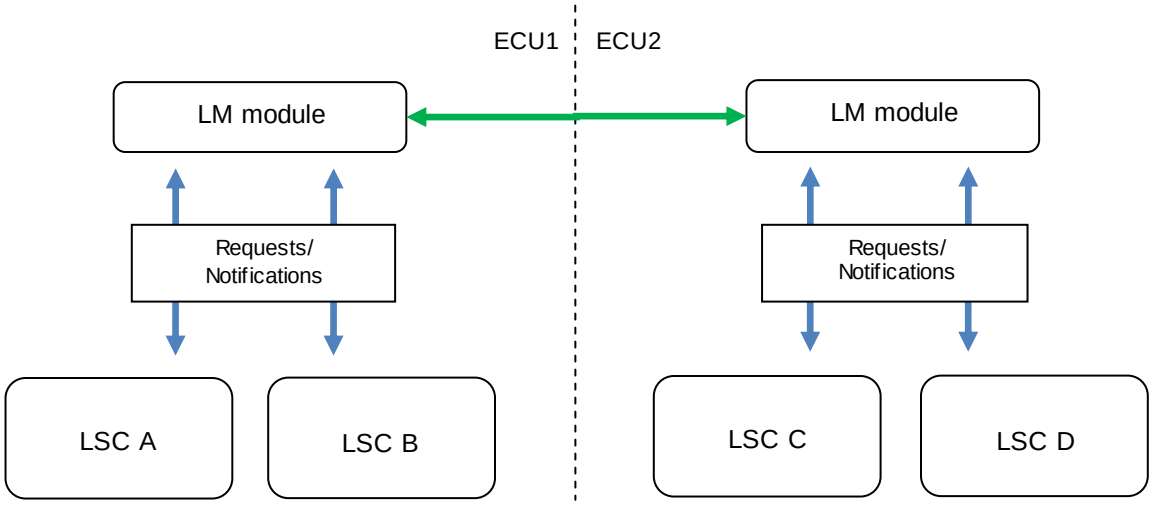



**Figure 32: VCC IP Link Manager – Overview of the communications path shown by the arrows. The green arrow is an IP based communications link while the blue arrows represent local SW interfaces.**

## 3.1.1.1 General Requirements

### 3.1.1.1.1 [VCC IP Prot: 0070/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define which VCC platforms shall use the VCC IP Link Manager. |
| Source | |
| Verification Method | Analysis |

**Req:** LM shall be used on the VCC SPA platforms.

### 3.1.1.1.2 [VCC IP Prot: 0071/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define which ECU's that shall deploy the LM module |
| Source | |
| Verification Method | Test |

**Req:** The LM module shall be deployed by all vehicle internal IP ECU's.

### 3.1.1.1.3 [VCC IP Prot: 0072/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the purpose of the LM module |
| Source | |
| Verification Method | Test |

**Req:** The LM module shall be responsible for the following tasks;

- Informing external LM modules across the vehicle internal IP network when the ECU has all its services available.
- Request services from other ECUs (meaning that the other ECUs should not shutdown).
- Handle service requests from external LM modules.
- Notify LSCs availability of services and external LM modules requests.
- Activate LM modules on external ECUs (by starting the ECUs) via the vehicle buses.

### 3.1.1.1.4 [VCC IP Prot: 0073/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define that LM messages shall use the VCC IP Command Protocol |
| Source | |
| Verification Method | Test |

**Req:** All LM messages shall use the VCC IP Command Protocol and shall have the following properties:

- ServiceID (16 Bits): 0xFFFF.
- OperationID (16 Bits): 0xFF01.
- OperationType (8 bits): 0x06 (`NOTIFICATION_CYCLIC`, not to be ACKnowledged)
- DataType (8 bits) : 0x01 (Meaning the signal should not be ASN.1 encoded)

### 3.1.1.1.5 [VCC IP Prot: 0074/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the LM (*IP_activity*) message structure |
| Source | |
| Verification Method | Test |

**Req:** The following LM header shall be used when constructing the *IP_activity* message, and shall be followed directly after the VCC IP Command Protocol Header.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VCC IP Command Protocol Header<br>(Figure 8: VCC IP Header Format on Application Layer) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **ACTION [8 bit]** | | | | | | | | **PRIO [8 bit]** | | | | | | | | **Reserved [16 bit]** | | | | | | | | | | | | | | | |

**Figure 333: VCC IP Link Management Header**

## Header fields:

**ACTION:**

| Values | Name | Description |
|---|---|---|
| 0x01 | AVAILABLE | Services on local ECU are available to use. |
| 0x02 | REQUEST_RG_1 | Request at least one service on Resource Group 1. |
| 0x04 | REQUEST_RG_2 | Request at least one service on Resource Group 2. |
| 0x08 | REQUEST_RG_3 | Request at least one service on Resource Group 3. |
| 0x10 | REQUEST_RG_4 | Request at least one service on Resource Group 4. |
| 0x20 | REQUEST_RG_5 | Request at least one service on Resource Group 5. |
| 0x40 | REQUEST_RG_6 | Request at least one service on Resource Group 6. |
| 0x80 | REQUEST_RG_7 | Request at least one service on Resource Group 7. |

Resource Groups, defined in 3.1.1.6.1 [VCC IP Prot: 0089/;-0]

**PRIO:**

| Values | Name | Description |
|---|---|---|
| 0x00 | PRIO_NORM | Normal priority. |
| 0x01 | PRIO_HIGH | When requesting a high priority keep-alive, meaning the transmitting ECU and ECUs receiving ECUs should ignore normal power handling. |

## 3.1.1.2 Service Discovery Transport protocol

### 3.1.1.2.1 [VCC IP Prot: 0075/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the LM protocol |
| Source | |
| Verification Method | Test |

**Req:** LM messages shall be based on VCC IP Command Protocol over **UDP**.

### 3.1.1.2.2 [VCC IP Prot: 0076/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the LM messages shall use the VCC IP Command Protocol |
| Source | |
| Verification Method | Test |

**Req:** LM messages shall be sent using the port defined in the "VCC IP Command Bus – SPA, doc nr: 31841985.

### 3.1.1.2.3 [VCC IP Prot: 0077/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the LM messages shall be sent among the ECUs. |
| Source | |
| Verification Method | Test |

**Req:** LM messages shall use the subnet-directed broadcast. The address is defined in REQPROD: 76545.

### 3.1.1.3 VCC IP Link Manager - ECU Initialization

#### 3.1.1.3.1 [VCC IP Prot: 0078/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the LM module's initial discovery mechanism. |
| Source | |
| Verification Method | Test |

**Req:** The LM module shall start broadcasting the message referred to as *IP_activity* periodically **once every second** after all local services have started. If all services could not be started, the LM module shall not broadcast *IP_activity* ([VCC IP Prot: 0074/;-0]).
The *IP_activity* message shall have the following default values in the header field;

ACTION = AVAILABLE
PRIO = PRIO_NORM

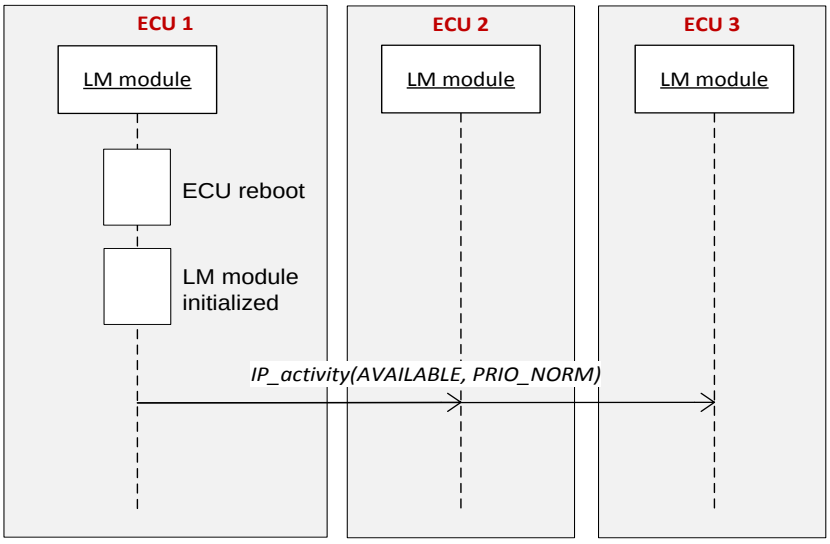



**Figure 34: Showing the LM module broadcasting service availability.**

#### 3.1.1.3.2 [VCC IP Prot: 0079/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the order in which services and the LM module shall be initialized at ECU startup. |
| Source | |
| Verification Method | Test |

**Req:** At ECU startup, the LM module shall be initialized after **all** preconfigured services have been initialized and ready to process received messages.

## 3.1.1.4 ECU Local

### 3.1.1.4.1 [VCC IP Prot: 0080/;-1]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define constants used locally by the LM module. |
| Source | |
| Verification Method | Test |

**Req:** The constants *Number_of_Nodes_in_RG_X*, where X is a number 1 – 7, shall be specified in a local configuration file.
*Number_of_Nodes_in_RG_*X is the number of vehicle internal IP ECUs that are expected to broadcast their availability for a specific Resource Group. It is used to conclude to what extent the vehicle internal IP network is available.

### 3.1.1.4.2 [VCC IP Prot: 0099/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define variables used locally by the LM module. |
| Source | |
| Verification Method | Test |

**Req:** The variable *Request_monitoring_timeout* shall be set for the LM module to use for monitoring the IP Links.
*Request_monitoring_timeout* represents the time it takes to declare an Active Request Session (defines in [VCC IP Prot: 0083/;-0]) dead. The *Request_monitoring_timeout* value shall be set larger than the broadcasting interval time (which is 1 second) to be sure that the *IP_activity* messages have time to be received and processed before the timer times out. The value shall be set larger the larger the network becomes.
*Request_monitoring_timeout* shall be calculated using the following equation.

**Request_monitoring_timeout = 3 + (0.25 * Numeber_of_Nodes_in _RG_*X)***

The value shall be calculated once for the given car configuration and shall be reused after that.

### 3.1.1.4.3 [VCC IP Prot: 0081/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define usage of the monitoring timers in the LM module. |
| Source | |
| Verification Method | Test |

**Req:** The LM module shall have one timer for each monitored IP Link. This timer is set to the value given by *Request_monitoring_timeout*. The timer shall start when the LM module has registered an ARS upon reception of the IP_activity message.
Each time LM receives an *IP_activity* message it shall restart the interface-specific timer, and when an interface is declared dead its timer shall stop.

### 3.1.1.4.4 [VCC IP Prot: 0082/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define which information the LM module shall provide to LSCs. |
| Source | |
| Verification Method | Test |

**Req:** It shall be possible for an LSC to be notified if the requested Service is *Available*, *Partly available* or *Unavailable*. Were

1. *Available* means that the *IP_activity* messages from all *Number_of_Nodes_in_RG_X* nodes have been received on the vehicle internal IP bus within the last *Request_monitoring_timeout* seconds. The number of expected messages (*Number_of_Nodes_in_RG_X*) depends on the requested Resource Group.
2. *Partly available* means that at least one *IP_activity* message ([VCC IP Prot: 0074/;-0]) has been received on the vehicle internal IP bus within the last *Request_monitoring_timeout* seconds.
3. *Unavailable* means that no *IP_activity* signals have been received on the vehicle internal IP bus within the last *Request_monitoring_timeout* seconds.

**Note:** An LSC may choose to try using the IP network even if all IP nodes are not visible on the vehicle internal IP network yet.

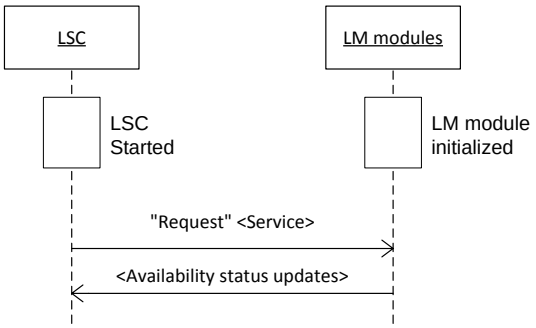



**Figure 35: Showing ECU local communication between an LSC and the LM module.**

### 3.1.1.4.5 [VCC IP Prot: 0083/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the term Active Request Session, ARS. |
| Source | |
| Verification Method | Test |

**Req:**
An Active Request Session, ARS is defined as either:

- There was at least one received *IP_activity* message, with ACTION-field = REQUEST_RG_X within the last *Request_monitoring_timeout* seconds.
- The signal *Vehicle_bus_ECU_wakeup* has been received (2.1.1.4.10) with property *RESOURCE* = *RG_X* but the LM module has not yet reached a point in time R*equest_monitoring_timeout seconds* after where it has started to send out the *IP_activity* message.
- There exist at least one service request made by a local LSC with PRIO-field = PRIO_HIGH

The ARS is interface specific, meaning there must exist one ARS towards each ECU to have a fully capable vehicle internal IP network.

### 3.1.1.4.6 [VCC IP Prot: 0084/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the interface between the Local Software Component, LSC and the LM module. |
| Source | |
| Verification Method | Test |

**Req:** The LM module shall offer Local Software Components, LSCs an interface (visualized in Figure 35) for determining whether there exists any ARSs and if any of them has the property PRIO_HIGH (explained in [VCC IP Prot: 0074/;-0]).
The LM module shall also offer information about the currently requested Resource Groups.

### 3.1.1.4.7 [VCC IP Prot: 0085/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how a LSC trigger a vehicle internal IP bus activation |
| Source | |
| Verification Method | Test |

**Req:** A Service Request from an LSC (Local Software Component) shall trigger a vehicle internal IP bus activation as specified by [VCC IP Prot: 0087/;-0].

### 3.1.1.4.8 [VCC IP Prot: 0086/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To state that an LSC shall be able to request to release the vehicle internal IP bus. |
| Source | |
| Verification Method | Test |

**Req:** An LSC shall be able to inform the local LM instance that the vehicle internal IP bus is not needed by the LSC anymore. Only if no other LSCs need the vehicle internal IP Bus, the LM instance shall release it according to [VCC IP Prot: 0088/;-0].

**Note:** This does not necessarily mean that the whole vehicle internal IP bus will shut down, since it may still be requested by other ECUs.

## 3.1.1.5 Bus activation requests

### 3.1.1.5.1 [VCC IP Prot: 0087/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how to request the vehicle internal IP bus |
| Source | |
| Verification Method | Test |

**Req:** When the vehicle internal IP bus is requested by an LSC with <*Prio*>, the LM module shall:

1. Check if the vehicle internal IP bus already is up, if it is go to step 3.
2. Request to activate the vehicle internal IP bus according to [VCC IP Prot: 0092/;-0].
3. Change parameters in the *IP_activity* message already being broadcasted to ACTION = *REQUEST_RG_X,* PRIO = highest currently requested <*Prio*>.

**Note:** Observe the example below, Example 1.

### 3.1.1.5.2 [VCC IP Prot: 0088/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how the LM instance releases the vehicle internal IP Bus. |
| Source | |
| Verification Method | Test |

**Req** When the vehicle internal IP bus is no longer needed, the LM instance needs to change the parameters in the *IP_activity* message being broadcasted to ACTION = *AVAILABLE,* PRIO = *PRIO_NORM (*[VCC IP Prot: 0074/;-0]*).*

**Note:** Observe the example below, Example 1.

### 3.1.1.5.3 Example 1

Scenario:

- LM on ECU 1 has been requested to keep alive the vehicle internal IP bus with priority <prio> which it will continue until requested not to.
- LM on ECU X has started broadcasting availability after start-up and will do so until not needed anymore or requested to itself keep alive the IP bus.

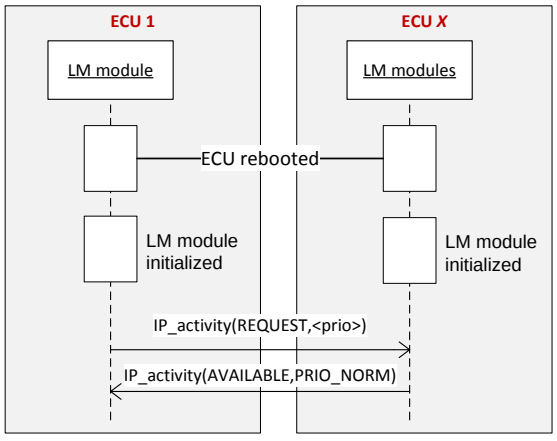



**Figure 36: Example showing the scenario when one LM instance Requests to keep alive the IP Bus while the other only broadcasts availability.**

## 3.1.1.6 Vehicle bus requests

### 3.1.1.6.1 [VCC IP Prot: 0089/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the usage of Resource Groups. |
| Source | |
| Verification Method | Test |

**Req:** The vehicle internal IP ECUs are assigned members of one or more Resource Groups, RG**.** The LocalConfig value *ResourceGroupMember* determines which Resource Groups the local LM module is a member of.

Decoding of *ResourceGroupMember* shall be performed according to the following table.

| Value | Name | Description |
|---|---|---|
| 0x01 | Reserved | Reserved for future use |
| 0x02 | RG_1 | Member of Resource Group 1. |
| 0x04 | RG_2 | Member of Resource Group 2. |
| 0x08 | RG_3 | Member of Resource Group 3. |
| 0x10 | RG_4 | Member of Resource Group 4. |
| 0x20 | RG_5 | Member of Resource Group 5. |
| 0x40 | RG_6 | Member of Resource Group 6. |
| 0x80 | RG_7 | Member of Resource Group 7. |

### 3.1.1.6.2 [VCC IP Prot: 0090/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how to wake up the vehicle internal IP bus via the vehicle bus |
| Source | |
| Verification Method | Test |

Multiple rows
Each row in the table below shall be handled individually. Only the deactivation criteria on the same row as the triggered activation criteria shall be able to deactivate the VFC.
Several activation criteria could potentially be triggered at approximately the same time, creating interleaved activations. All deactivation criteria's on the triggered rows have to trigger for the VFC to deactivate in these scenarios.
If there are multiple activation requirements on the same VFC, then all of these requirements shall be handled in the same way; individually.

Timers
Deactivation criteria consisting of a VFCTimeOutDelay indicate that each VFC activation shall reset a timer called VFCTimeOutDelay. The VFCTimeOutDelay is false until the timer reaches its preset value (3 seconds unless otherwise specified in the table) when it turns to true. The VFC shall be deactivated when the entire expression is true.

| LC | On Criteria | | Off Criteria | |
|---|---|---|---|---|
| VCC IP Link Manager | Signal/Timer | Description | Signal/Timer | Description |
| | An LSC (Local Software Component) requests a remote service<br>AND<br>IP interface to the service providing ECU(s) is not active. | Wakeup of the IP link between the local and the remote ECU. | VFCTimeOutDelay = 3 sec | Time out after 3 seconds |

### 3.1.1.6.3 [VCC IP Prot: 0091/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define the content of the vehicle bus signaling |
| Source | |
| Verification Method | Test |

**Req:** A V*ehicle_bus_ECU_wakeup* signal (Flexray, CAN, LIN) with content "<*Prio, ResourceGroup*>" shall be used as wakeup signal when the vehicle internal IP bus is not available. Each ECU will publish its own signal. The actual V*ehicle_bus_ECU_wakeup* from each vehicle internal IP ECU is specified in the Infotainment Platform SRD.
During multiple simultaneous requests, <Prio> parameter shall always be set to PRIO_HIGH if any request has <Prio> = PRIO_HIGH.

**<ResourceGroup> - Parameter**

| Value | Name | Description |
|---|---|---|
| 0x01 | Reserved | Reserved for future use |
| 0x02 | RG_1 | Request at least one service on Resource Group 1. |
| 0x04 | RG_2 | Request at least one service on Resource Group 2. |
| 0x08 | RG_3 | Request at least one service on Resource Group 3. |
| 0x10 | RG_4 | Request at least one service on Resource Group 4. |
| 0x20 | RG_5 | Request at least one service on Resource Group 5. |
| 0x40 | RG_6 | Request at least one service on Resource Group 6. |
| 0x80 | RG_7 | Request at least one service on Resource Group 7. |

**<Prio> - Parameter**

| Name | Value |
|---|---|
| PRIO_NORM | 0 |
| PRIO_HIGH | 1 |

Default values are *Prio = PRIO_NORM* and no *ResourceGroup* is requested, which means that the vehicle internal IP network is not requested.

### 3.1.1.6.4 [VCC IP Prot: 0092/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how to wake up the vehicle internal IP bus via the vehicle bus |
| Source | |
| Verification Method | Test |

**Req:** How to request wakeup of the vehicle internal IP bus via the vehicle bus signaling

1. VFC "*IP Wakeup*" shall be activated, see [VCC IP Prot: 0090/;-0]
2. *ResourceGroup = RG_X, Prio =* <*Prio*> shall be set in the V*ehicle_bus_ECU_wakeup* signal as long as VFC *"IP Wakeup"* is activated, and then return to default values as specified in [VCC IP Prot: 0091/;-0].

### 3.1.1.6.5 [VCC IP Prot: 0093/;-0]

| Attribute name | Attribute value |
|---|---|
| Purpose | To define how to handle the vehicle bus ECU wakeup signal |
| Source | |
| Verification Method | Test |

**Req:** When the property *ResourceGroup = RG_X* is detected on the vehicle bus ECU wakeup signal in [VCC IP Prot: 0091/;-0]), and the *X* in *RG_X* refers to the Resource Group number which the LM module is a member of according to [VCC IP Prot: 0089/;-0], the following shall happen:

1. Initialize the vehicle internal IP bus according to 3.1.1.3, once the ECU has started.

# 4. Appendix A - asn.1 EXAMPLES

## 4.1 Introduction to using the generated asn.1 code

This section describes how to use code generated by the proposed ASN.1 compiler. It should however be possible to use any asn.1 compiler.

The ASN.1 description:

```
ContextNumber ::= SEQUENCE
{
                                                originFromTem      BOOLEAN DEFAULT
                                                false,
                                                cSequenceNr  INTEGER (0..7),
                                                mSequenceNr  INTEGER (0..7),
                                                moreFlag     BOOLEAN DEFAULT false
}

The result in the header file for C-code:
typedef struct ASN_ContextNumber_t * ASN_ContextNumber;

struct ASN_ContextNumber_t
{
    BOOL originFromTem;
    U32 cSequenceNr;
    U32 mSequenceNr;
    BOOL moreFlag;
};
U32 ASN_ContextNumber_EncodedSize( ASN_ContextNumber ThisPtr );
#ifdef ENABLE_ASN_SESSION_SIZE
  U32 ASN_ContextNumber_SessionSize();
#endif /* #ifdef ENABLE_ASN_SESSION_SIZE */
ASN_Result ASN_ContextNumber_Decode( ASN_ContextNumber ThisPtr, ASN_Session
                                                session,  ASN_Stream stream);
ASN_Result ASN_ContextNumber_Encode( ASN_ContextNumber ThisPtr, ASN_Stream
                                                stream);
ASN_ContextNumber ASN_ContextNumber_Create( ASN_Session session);
#ifdef ASN_PRINT
  void ASN_ContextNumber_Print( ASN_ContextNumber ThisPtr );
#endif /* #ifdef ASN_PRINT */
```

### 4.1.1 The structure

The primitive types are accessible as members of the struct. The exception is the OctetString, which is used to hold a string or an array of bytes. To access this member you must use specific functions. Set, SetText, Get, GetAsString, Associate, and AssociateText.

### 4.1.2 The create function

The create function will allocate the struct using memory from the session. All primitive types are initialized with zero or the default value if existing. The non-primitive types will be initialized using their own create function. The return value is a pointer to the new structure if successful or a null pointer if the allocation fails.

### 4.1.3 The encoded size function

This function returns the number of bits the structure, and its possible contained structures, will need to be encoded onto a stream. This value is based on the data in the structures.

### 4.1.4 The session size function

This function returns the maximum number of bytes this structure could need from the session when decoded. This information is not based of the contained data. This function must be enabled by the define ENABLE_ASN_SESSION_SIZE.

### 4.1.5 The decode function

This function will fill the members of the data structure with values from the stream. It will create contained structures as needed. The function will return ASN_RESULT_OK or else an ASN_Result pointer containing the error code and the line-number in the generated code. The possible error-codes are ASN_ERROR_MEMORY, ASN_ERROR_BUFFER_STARVING.

### 4.1.6 The encode function

This function will encode the values from the members onto the stream. The function will return ASN_RESULT_OK or else an ASN_Result pointer containing the error code and the line-number in the generated code. The possible error-codes are ASN_ERROR_BUFFER_FULL, ASN_ERROR_VALUE_NOT_WITHIN_RANGE, ASN_ERROR_SIZE_NOT_SUPPORTED.

### 4.1.7 The print function

This function prints the content if the data structure.
To enable this function you must define ASN_PRINT and implement three functions declared in the asn_base.h file: `ASN_PrintSetIndent`, `ASN_Print`, `ASN_PrintIndented`
The result of the print-function looks like this:

```
contextNumber
{
  originFromTem
  {
    true
  }
  cSequenceNr
  {
    6
  }
  mSequenceNr
  {
    0
  }
  moreFlag
  {
    false
  }
}
```

### 4.1.8 The types

#### 4.1.8.1 The built-in types – the primitives

#### 4.1.8.2 INTEGER

Must have a constraint defining the size used. This is done with a min and max values. Integer is always transferred as a unsigned number as "the actual number" – "the min value". Both min and max could be negative.

#### 4.1.8.3 ENUMERATION

This is handled as an integer but with defined names. It could have a size constraint, but would otherwise be as small as needed, depending on the number of values declared.
Tip: To improve readability of a packed message it's often a good idea (if minimal packed size is not a goal) to aim for byte aligned sizes, i.e. to constrain the enumeration to (0..255)

#### 4.1.8.4 BOOLEAN

One bit of data, that's all.

#### 4.1.8.5 OCTET STRING

An array of bytes, that always has a size constraint. The size is always coded directly in-front of the byte array. This is the closest thing to a string, and this compiler always decodes this data onto a buffer with one extra byte that is zero. It's thereby accessible as a string even though it might not be printable characters inside.
An OCTET STRING member must be manipulated using the specific ASN_Octet_String functions and is not accessible as direct as the other primitive types.

#### 4.1.8.6 UTF8String

UTF8String is in essence the same as an OCTET STRING, but where the name indicates the type of content.

#### 4.1.8.7 NULL

NULL is a special type indicating that an object exists but has no data. Mostly used in CHOICE constructs.

#### 4.1.8.8 Ellipsis Extensibility Marker "..."

The ellipsis "..." is mostly used as the last element in a SEQUENCE, where it indicates that the sequence may later be extended with more elements. The construction makes it possible for an extended message to be backwards compatible with receivers who do not know about the extension. Any new elements must then be added after the extension mark.

*OBS: MSWord often autoconverts the three dots "..." into the single character "…" which will not be understood by the code generator.*

### 4.1.9 Make your own types – your building blocks

The type-name of any defined ASN.1 object must start with a capital letter. The name of the elements within a defined type must start with a lowercase character.

#### 4.1.9.1 SEQUENCE – The struct

This is a simple "container class". The elements are coded in the order they are stated in the definition, but the sequence in itself does not encode to any data. I.e. it is only a convenient way of collecting several other elements.
Elements could be OPTIONAL. The standard of PER encoding states that the needed flags (one bit each) should be coded at the beginning of each sequence. This compiler makes it possible to place the flag wherever needed; as long as it's before the element it referees too.

#### 4.1.9.2 CHOICE – The union

Only one of the elements inside a choice will be encode/decoded, based on the choice. The choice value (or type field) is always encoded first followed by the chosen element. There is a size constraint on the choice declaring how many bits to use for the choice field and thereby how many different elements you can have. In the standard there are no way of declaring the value of each choice, but that is added to this compiler. Otherwise you would be forced to declare 255 NULL values to be able to handle a message type filed of one byte and declaring message number 255.

### 4.1.9.3 SEQUENCE OF – The array

This type declares an array of either another defined type, or an array of any build-in types.

## 4.1.10 How-To

The dummy examples below assume the ASN.1 definition:

```
GatsMessage::=SEQUENCE {
     Message    CHOICE{
        tnReq  SEQUENCE {
           latitude   INTEGER,
           longitude  INTEGER
        }
     }
}
```

The code is taken from a more complex example and may not compile.

### 4.1.10.1 Decode

You must know the type of message in your buffer, i.e. the top-level structure (root object) to be used when decoding.

- Create a session, big enough to receive all structures. Use the SessionSize function for the root object as a maximum size needed.
- Create a Stream and associate your data buffer with this stream.
- Create an instance of the root object using the corresponding Create function.
- Decode the stream using the Decode function for the root object.
- Accessing the content using normal struct member access. Be sure to check the type of every choice type and the size of every array to be able to access data within these structures.

#### 4.1.10.1.1 Code example

```
void test_Decode( void )
{
   ASN_BYTE struct_buffer[2000];
   ASN_BYTE encode_buffer[ASN_STREAM_SIZE];

/*A simulated incoming message in text format*/
char * sMessage =
"221F00434090000001008081715A0B0FC08A3232323232323232323232323232323232300302805FFE74
8000A93130000FFD6085FB13A736E65000000000000C01042FE4FF39B08303605B1C22021F08217F26
D9CD86203289D7C09000F8410BF93DCE6C4DC1AC46AE348147C2085FC65A7362C7034050D8012A3E10
42FE2D739B17D02405A807C593F00";

   ASN_BYTE byMsg[140];
   U32 iOriginalBufferSize = HexToByte( sMessage, byMsg );
   ASN_Result pResult;
   GatsMessage pMsg; //Top level ASN type
```

```
   ASN_Session session = ASN_Session_Create( struct_buffer, sizeof( struct_buffer
) );
   ASN_Stream stream = ASN_Stream_Create( encode_buffer, sizeof( encode_buffer )
);

   ASN_Print( "TN Request start" EOL );

   ASN_Stream_AttachBuffer( stream, byMsg, iOriginalBufferSize );

   pMsg = GatsMessage_Create( session );

   pResult = GatsMessage_Decode( pMsg, session, stream );

   if ( pResult != ASN_RESULT_OK )
   {
      ASN_Print( "Decode failed at line: %d, Error: %d" EOL, pResult->linenumber,
pResult->error );
      /*
      ASN_GatsMessage_Print( pMsg );
      */
      return;
   }
   else
   {
      /*
      ASN_GatsMessage_Print( pMsg );
      */
   }

   /* Get the data elements in the message. "GatsMessage" starts with a CHOICE.
You can't access into a choice before checking the type. */
   if ( pMsg->message->choice == e_MessageType_tnReq )
   {
      S32     s32Lat = pMsg->message->choice->latitude;
      S32     s32Long = pMsg->message->choice->longitude;


   }
   ASN_Print( "TN Request end" EOL );
}
```

## 4.1.10.2 Encode

- Create a session, big enough to contain all structures. Use the SessionSize function for the root object as a maximum size needed.
- Create an instance of the root object using the corresponding Create function.
- Fill all members of the data structures; be sure to set the type of every choice and setting the size of every array before setting values inside these structures.
- Create a Stream and associate your data buffer with this stream.
- Encode the data onto the stream using the Encode function of the root object.
- Align the buffer to a byte boundary. This will ensure that the unused bits will be filled with zero. Most layered protocol demands a byte alignment between the layers.
- Retrieve the data buffer using the ASN_Stream_GetBuffer, or at least getting the size coded if you associated your own buffer, using the same function.

`Code example` This example encodes a « Gats Message »

```
void test_Encode( void )
{
   ASN_BYTE decoded_buffer[ASN_SESSION_SIZE];
   ASN_BYTE encoded_buffer[ASN_STREAM_SIZE + 140];
   ASN_BYTE * au8ResultData;
   U32 iResultDataSize;
   ASN_Session session = ASN_Session_Create( decoded_buffer, sizeof(
decoded_buffer ) );
   ASN_Stream stream = ASN_Stream_Create( encoded_buffer, sizeof(encoded_buffer)
);
   GatsMessage pMsg; //The top level ASN type
   ASN_Result pResult;

   pMsg = GatsMessage_Create( session );


      /*
   * The "message type" must be set in this case (because it's a choice),
   * and thereby creating the underlying data structures.
   */
   MessageType_SetType( pMsg->message, session, e_MessageType_tnReq );

   //Set some values in the message
   pMsg->message->tnReq->version = 2;
   pMsg->message->tnReq->appIdentifier = e_ApplicationIdentifier_tnService;
   pMsg->message->tnReq->initiativeFlag = true;


   /*
   * This way of setting a text is consuming memory from the session and
   * copies the information.
   */
   ASN_OctetString_SetText( pMsg->adp->message->tnReq->vmasterdata->vin->text_8,
session, "FFFFFFFFFFFFFFFF" );


   /* Now encode the message */
   pResult = GatsMessage_Encode( pMsg, stream );

   if ( pResult != ( ASN_Result)ASN_RESULT_OK )
   {
      ASN_Print( "Failed to encode the Gats message. Error:" EOL "%d at line %d"
EOL, pResult->error, pResult->linenumber );
      return;
   }

   ASN_Stream_AlignOnByte( stream );
   au8ResultData = ASN_Stream_GetBuffer( stream, &iResultDataSize );
```

## 4.2 Appendix - Traditional Datatypes

Traditional datatypes are supported and may be used:

| Type | Description | Size [bit] | Remark |
|---|---|---|---|
| Boolean | TRUE/FALSE | 8 | FALSE (0x00), TRUE (0x01) |
| Uint8 | Unsigned integer | 8 | 0 to 255 |
| Uint16 | Unsigned integer | 16 | 0 to 65,535 |
| Uint32 | Unsigned integer | 32 | 0 to 4,294,967,295 |
| Sint8 | Signed integer | 8 | -128 to 127 |
| Sint16 | Signed integer | 16 | −32,768 to 32,767 |
| Sint32 | Signed integer | 32 | −2,147,483,648 to 2,147,483,647 |
| Float32 | Floating point number | 32 | IEEE 754 binary32 (Single Precision) |
| Float64 | Floating point number | 64 | IEEE 754 binary64 (Double Precision) |
| String | UTF-8 | | |

**Table 11: List of supported data types**

### 4.2.1 Range/Sign to Data Type

| Range/Sign | Data Type | Description |
|---|---|---|
| 0x00 … 0xFF | Uint8 | - |
| 0x00 … 0xFFFF | Uint16 | - |
| 0x00 … 0xFFFFFFFF | Uint32 | - |
| [ … ] | Byte array | Array of consecutive bytes |
| String | UTF-8 | UTF-8 according to chapter 4.2.9 & 4.2.10 |

**Table 12: Map data range/sign to data type**

### 4.2.2 Byte

1 byte (8 bits).

### 4.2.3 Word

2 byte (16 bits).

### 4.2.4 Long

4 bytes (32 bits)

### 4.2.5 Boolean

1 byte (8 bits). Only one bit of the byte can be used.

0x00 = False
0x01 = True

### 4.2.6 Enum

1 byte. Used to represent values across a series of named constants. Enums contains unchangeable number of invariable elements.

E.g range of values: 0x00 … 0x04.
0x00: value one
0x01: value two
0x02: value three
0x03: value four

### 4.2.7 Bitfield

With a bitfield, it's possible to either set bits or, clear bits using OR, AND NOT, XOR
Bitfields are divided into two parts: Mask and Data. “Mask” is a masking bit field of the same size as the Data Content Area “Data”.

Size: The total size of the bitfield is either 1,2,4 or 8 bytes.
Data: ½ size byte (Data content area)
Mask: ½ size byte (Masking area)

Example of a bitfield: (XXXX XXXX **,** YYYY YYYY), where X represents Mask and Y represents Data.

### 4.2.8 Sequence

Individual parameters in sequence does not to have to be of the same type. E.g. <parameter>, <parameter>, <parameter>

### 4.2.9 String (fixed length)

Strings are encoded using Unicode and are terminated with a "\0" character. The length of the string (this included the termination character) in Bytes has to be specified in the interface definition. Fill in unused space using "\0". Unicode encoding shall UTF-8.

### 4.2.10 String (dynamic length)

Strings with dynamic length start with a length field [32 Bit] in the data payload. The length is measured in Bytes and is followed by "\0"- terminated string data. The interface definition must also define the maximum number of bytes the string (including termination with "\0") can occupy. Unicode encoding shall UTF-8.

### 4.2.11 StaticArray (fixed length)

#### 4.2.11.1 Onedimensional Array

One-dimensional array (single dimensional) with a fixed length carries n number of elements of the same type.

StaticArray [n]

#### 4.2.11.2 Multidimensional Array

Multidimensional arrays are an "array of arrays". One array [n] element contains another array with elements [m].

StaticArray [n][m]

### 4.2.11.3 Dynamic Array (dynamic length)

Arrays with dynamic length start with a length field [32 Bit] in the data payload. This means that the dynamic length size is added in front of the actual data. Dynamic arrays can either be one-dimensional or multidimensional.