# Technical description for the communication between BlueRange APP and BlueRange Node

## Inhalt

# 1. History

| Date | Version | Page | Description |
|---|---|---|---|
| 2017 Braunschmid(Br), Zehender (Ze) | 1..7 (draft) | init | Initial versions |
| 21-09-2017, Niemann(Ni) | 8 (draft) | Appendixes | Changed Para and response scheme on some commands, added new command '*NodeDeleteVirtualLum'*, changed '*update_level' to* 16-bit |
| 27-09-2017, Ni | 9 (draft) | Appendix B | A system time has no ID since it is generic |
| | | Appendix E | Parameter change Event message |
| | | Appendix K | Event and action type combination, preliminary |
| 05-10-2017, Ni, Ze | 9.1 | Appendix N,O | Reject "draft", added CRC8 and Encryption (XTEA) description |
| 05-10-2017, Ze | 10 | | Included 'Requirement for Bluerange Mesh' |
| 15-10-2017, Ze | 10.1 | Appendix P,Q | Sequence Table |
| 16-10-2017, Ni, Ze | 10.2 | | Implementation details, corrections |
| 19-10-2017, Ni, Ze | 11 | | New action types AT_START_SEQUENCE, AT_STOP_SEQUENCE,<br>new event type ET_TIMER, ET_REGULATION and their respective commands,<br>Added Status byte in *SetNodeVirtualLumData* |
| 23-10-2017, Ze | 12 | | On m-way request, changed some formulation in requirements |
| 25-10-2017, Ze | 12.1 | | More formulation changes for m-way |
| 26-10-2017, Ze | 13 | | Better specification for HCI mode |
| 26-10-2017, Ze | 14 | | Removed '….resulting from IP infringement claims.'. |
| 15-11-2017, Ni | 14_1 | Appendix B,C,I | Appendix B:<br>Response of command *NodeCreateVirtualLum* has been adjusted;<br>Command *Set/GetNodeVirtualLumData* contains passive / active etc. brightness % values instead levels: has been adjusted to one byte. This is because of dimming processes of RGBWAF lightLevel channels which are based and calculated via brightness % values;<br>Deletion of a physical lum channel in a virtual luminaire: Byte 1, bit 7 of command *SetPhysLumChannel* has been adjusted to do so;<br>Introducing command *GetVirtLumLevel*. Helpful for slider sync on HandyApp;<br>Appendix I:<br>Some MSG_STATUS_xxx has been added;<br>Chapter 6.5:<br>Corrected log10 formula;<br>Command *GetColorControl* and *SetNodeScene*: value range of brightness is 0…254<br>Appendix C:<br>Light Reaction Dim Brightness / Color has been adjusted; max. ActionTyp Parameter 6 |

| | | | |
|---|---|---|---|
| 24 -11-2017, Buelow | | | Spelling corrections and improvements (corrections of drawings).<br>Added M-Way support for testing a DUT in the tester in chapter "Requirements on BlueRange Mesh. |
| 27-11-2017<br><br>27-11-2017, Ni<br>28-11-2017, Ze<br>28-11-2017, Br<br>28-11-2017, Ni, Ze | | <br><br>Appendix B<br><br>4. Introduction<br><br>6.4.1<br>6.3<br>Appendix K<br>Appendix C<br>Appendix E | Moved the added informations of "M-Way support for testing a DUT in the tester" from 24-11-2017 to a new chapter "Appendix S"<br>Added *SetScene2VirtLum*<br>Added *GetSensorData*<br>Document Titel<br><br>TeachIn procedure must be approved by VS.<br>Command changed<br>Reworked<br>ActionType_VirtLumChannelSet, Parameter changed<br>Rejected ET_LIGHT_REGULATION, rejected PassiveArea Action |
| 07-12-2017, Ni | 15 | Appendix B | Added description for command requirements<br>changed PacketID size from 2 Bytes to 1 Byte<br>changed syntax of AT_LIGHT_REGULATION in AT_LIGHT_REGULATION_MODE |
| 14-12-2017, Ni<br><br>19.12.2017, Ni<br>20.12.2017, Ni<br><br>02.01.2018,Ni<br>11.01.2018,Ni<br><br><br><br>24.01.2018,Ni<br>24.01.2018,Ni | 16<br><br><br><br><br><br><br><br><br><br><br>17 | Appendix E<br><br><br>Appendix C<br>Appendix E<br><br>Appendix B<br><br><br><br><br>Appendix E,P<br>Appendix B | added response to an EventMsg<br>justify description for ET_LIGHT_LEVEL, Para 0<br>corrected possible values for virtLumNo[1..16]<br>ET_REGULATION: added initLR=4 for initialisation of a foreign node with several virtLum's who joins light regulation for light outputs<br>*SetScene2VirtLum* : added possible para val<br>Event-Msg will response as a UART-Response only<br>Action „virtLumChannelSet" now with „param5=start Scene". Allows assignment of lightlevel to a virtLum at light regulation start<br>Added a "Note" for fallback scene 1<br>Adjusted scene sequence<br>Added "In general" descriptions of command parameter |
| | 18 | Appendix B<br><br><br>Appendix E | NodeIdentifyPhysLum: added para addr val 0xfffe for deviceLED<br>SetNodeSystemKey: Parameter SystemKey "0" resets all node data<br>New Event ET_POWER_READY, which will be sent by VSfirmware |
| 26.01.2018, Ze<br>31.01.2018, Ni<br>02.02.2018, Ni | 19 | Appendix R<br>Appendix C<br><br><br>Appendix B<br><br>Appendix T | Added a table with possible Timer entry data combinations<br>Added descriptions for action types<br>Changed ActionType Param: now 5 parameter<br>Added "Manual mode" for AT_LIGHT_REGULATION_MODE<br>SetSeqEntry Param Byte 3 and 4 put together in byte 3 to save memory<br>SetNodeReactionEntry: added "user" possible values |

| | | | |
|---|---|---|---|
| | | Appendix E | Added Start up (Power on) behaviour<br>Added notes, e.g. Encryption type is UserKey if send from HandyApp, otherwise SystemKey |
| 07.02.2018,Ni | 20 | Appendix B | Set-/GetInput: Added value range for Byte 0<br>SetNodeVirtualLumData: Byte 3 “state” has been declared as free and will be ignored in VSfirmware<br>SetNodeVirtualLumData: dimSpeed in units of 700ms<br>Add a general note<br>GetNodeVirtLumData: byte 3 renamed in “modeState”<br>GetNodePhysLums: added Status info, adjusted response data<br>SetColorControl: jetzt zusätzlich mit UserKey Verschlüsselung |
| 14.02.2018,Ni | | Appendix I<br>Appendix P | GetNodeStatus: corrected description for byte 5<br>Added / corrected MSG_STATUS 0x13..0x19, 0x20,0x21,0x22,0x23,0x24,0x80<br>SceneSequence STOP interpretation 0x7E<br>Corrected Parameter for AT_SEQUENCE_PROCESS |
| 19.02.2018, Ni | | Appendix E<br><br>Appendix C | Rejected ET_SEQUENCE_PROCESS<br>Added notes, adjusted event nums<br>Corrected “numPara” |
| 07.03.2018, Ni | 21 | Appendix U<br>Appendix I<br>Appendix B | virtLum data explanation<br>added more MSG_STATUS_xx<br>added a general note: **If a rxd (uart) message frame fails: it may not possible to select between a command and an event. In this case a simple One-byte-uart response will be delivered with an appropriate MSG_STATUS content.**<br>Attached a Note to min Brightness |
| 16.03.2018, Ni | | Appendix C<br>Appendix E | Added AT_SLIDER_CONTROL<br>Added ET_SLIDER_CONTROL |
| 26.03.2018, Ni | 22 | Appendix B<br><br>Appendix V | Set-/GetVirtualLumData: replaced “light regulation active” with “moveSenseAct”<br>SetTimer variable “timestamp” now renamed in “time”<br>Added movement sensor action behaviour, moveSenseAct |
| 03.04.2018, Ni | | Appendix W<br>Appendix R | Added search mode behaviour<br>Added detail “SetTimer” examples |
| 05.04.2018, Ni | 23 | Appendix E<br>Appendix X<br>Appendix C<br><br>Appendix B<br>Appendix I | Changed behaviour of ET_BT_BUTTON parameter inputNum always 0<br>Added slider type definitions<br>Added para “sliderType” for AT_SLIDER_CONTROL<br>Renamed AT_LIGHT_REGULATION_MODE in AT_LIGHT_CONTROL<br>Added command `GetNewSearchData`<br>Added MSG_STATUS_PHYS_LUM_ADDR_ALREADY_USED 0x28<br>Renamed MSG_STATUS_INVALID_PHYS_LUM_NUM in *_PHYS_LUM_ADDR |
| Ze | | Appendix E<br>Appendix C<br>Appendix L | Added new event type ‘ET_EXT_BEACON’ for handling of external beacons.<br>Added new action Type ‘AT_BEACON_STATE’ to switch internal beacon function On/Off/Toggle.<br>Added command ‘SetBeaconState’ (together with ‘AT_BEACON_STATE’). |

| | | | |
|---|---|---|---|
| | | Appendix D<br>Appendix M | Added new Node type 'NT_VS_CONNECT_ME'.<br>Corrected BT button handling according inputNum parameter (always 0) |
| 10.04.2018, Ni | 24 | Appendix E<br>Appendix<br>B,C,U,V<br>Appendix P | Added comments for EventType, description for ET_CHANGE_SCENE modified<br>moved "moveSenseAct" to AT_LIGHT_CONTROL<br>added further explanation |
| 20.04.2018, Ni | 25 | Appendix B<br>Appendix C<br>Appendix E<br>Appendix D | Corrected SetNodevirtLumData, Added more explanations for SetInput Byte4,5<br>Renamed AT_VIRT_LUM_CHANNEL_SET into AT_LIGHT_REGULATION, reduced parameter<br>Added more explanations regarding eventTab & reactionTab<br>Added more explanations for node types |
| 04.05.2018, Ni | 26 | Appendix B<br><br><br><br><br><br>Appendix K<br>Appendix R<br>Appendix C<br>Appendix I | Added a general node regarding "Scene"<br>Command TransmitFirmwareData: Block size has been set to 256<br>Added info ET_SYSTEM_TIME: unused for now<br>SetTimer: added enable/disable flag<br>Changed Note: tab overwriting possible<br>Command and Response of `NodeCreateVirtualLum` has been redesigned<br>changed Event / Action Matrix<br>added an enable/disable flag for TimerTab, renamed "Repeat Field" in "Control Field"<br>changed parameter configuration for AT_TIMER<br>added MSG_STATUS_OVERWRITE_WARNING<br>distinguish between errors and warnings (>=0xE0) |
| 07.05.2018, Ni | 27 | Appendix B<br><br><br><br>Appendix K<br>Appendix Y | Added GetNodeStatus Response Status: Bit1<br>**Added a Note: The response of a command leads up with the commandNo always and follows the MsgStatus with Byte 0**<br>Rejected combination ET_TIMER with AT_DIM_UP_DOWN…<br>Added "Bootloader" info |
| 14.05.2018, Ni | 28 | Appendix B<br><br><br><br>Appendix I | Added array count in response of GetNodeVirtLums<br>Added "ID" in name of encryption type in address field<br>`Set-/GetNodeReactionEntry`: changed interpretation of Byte7, now UserkeyID<br>Added 6 new MSG_STATUS* |
| 23.05.2018, Ni, Ze | 29 | Appendix I<br><br>Appendix Z<br>Appendix B<br>Appendix Y | Added / Corrected new MSG_STATUS* regarding firmware update and other commands.<br>Added detail description of MSG_STATUS<br>Added description of VS internal commands for tester<br>Added a note to `NodeIdentifyPhysLum`, added new command "`Debug`"<br>Update VSfirmware download |
| 05.06.2018, Ni | 30 | Appendix I<br>Appendix Y | Added `MSG_STATUS_PREMIUM_FAILED, MSG_STATUS_EVENT_TAB_ERR`<br>Update VSfirmware download format description |

| | | | |
|---|---|---|---|
| 13.06.2018, Ni | 31 | Appendix B<br><br><br>Appendix I | VSfirmware download commands always with node key encryption<br>Added new command “GetVersion”<br>Changed a parameter of command `NodeIdentifyPhysLum`<br>edited `MSG_STATUS` |
| 20.06.2018, Ni | 32 | Appendix D<br>Appendix B | Changed wording<br>Added MSG_STATUS_RXD_PACKET_ERR<br>Changed Note: **The response of a command leads up with the commandNo, following MsgStatus, following 16-Bit updateLevel (MSB first, LSB second)**<br>Cmd “GetNodeUpdateLevel” has been rejected |
| 25.06.2018, Ni | 33 | Appendix B<br><br><br>Appendix D | Cmd “GetNodeUpdateLevel” has been re-integrated<br>Cmd “GetVersion” now with SystemKey encryption<br>Response of “GetNodeStatus” includes max. reactionTab entries in addition<br>changed description NodeType |
| 26.06.2018, Ni<br><br><br><br><br>28.06.2018, Bue | 34 | Appendix M<br>Definitions<br>Appendix C<br>Appendix B<br>Appendix I<br><br>Appendix E<br>Kap.6.7 | Add explanations: use case of light control and regulation<br>Add definition for a “scene”<br>Add param5=sceneNo for AT_LIGHT_CONTROL<br>Response auf GetNodeStatus liefert auch Anzahl möglicher User (maxUser)<br>Added MSG_STATUS_LIMIT_REACHED<br>Changed description `MSG_STATUS_OVERWRITE_INFO`<br>EV_BT_BUTTON_X can have variable inputNum[0..255]<br>Correct wording virtual channels in “virtual luminaires” |
| 04.07.2018, Ni | 35 | Appendix B<br><br>Appendix C<br>Appendix I<br>In general<br><br>Appendix E<br>Appendix A | SetNodeReactionEntry has UserkeyID 2-251,<br>changed SetInput / GetInput<br>AT_TIMER has been rejected<br>Added MSG_STATUS_DECRYPTION_FAILED, MSG_STATUS_WRONG_USER_KEY_ID<br>Changed ET_BT_BUTTON_x to ET_BT_BUTTON_TYPE_x in various descriptions. Instances of this type works thru “inputNum”…^<br>Changed #defines of all Events !<br>DestinationType has been defined |
| 10.07.2018, Ni | 36 | Use cases<br>Appendix B<br><br>Appendix T<br>Appendix F | BT Button and destinationType<br>SetUser pos can be delete thru 0xff<br>SetScene2VirtLum(0xFF,SceneNum) allows settings a LLScene to all available virtLums<br>Detail description StartUp of deviceNode<br>Detail description of packetType, destinationType |
| 16.07.2018<br>24.07.2018 | 37 | Appendix B<br>All<br>Appendix B | changed GetInput-Response (added “max Instanz”)<br>changed wording: BTfirmware=”VSfirmware”, BRfirmware=”MWAYfirmware”<br>Command Debug: changed Debug type 3 (re-program DALI-Groups) |

| | | | |
|---|---|---|---|
| | | Appendix C | AT_SLIDER_CONTROL added LLScene |
| 04.09.2018, Ni | 38 | Appendix C<br>Appendix X<br>Appendix K | AT_LIGHT_SCENE added param4 “mode”<br>New SLIDER_TYPE_LIGHT_REGULATION defined<br>Added combination ET_BUTTON←→AT_LIGHT_REGULATION<br>Added combination ET_ SLIDER_TYPE ←→AT_LIGHT_REGULATION |
| 10.09.2018, Ze<br>11.09.2018, Ni<br>14.09.2018, Ze | 39 | Appendix E<br>Appendix B<br>Appendix I | Added ET_IO_BUTTON parameter description to table<br>Added new Command SetBrightnessRef<br>Added MsgStautus MSG_STATUS_WRONG_NODE_TYPE 0x43 |
| 27.09.2018, Ni<br>05.10.2018, Ni<br>08.10.2018, Ni<br>18.10.2018, Ni | 40 | Appendix K<br>Appendix I<br>Appendix P<br>Appendix B | Some more explanations regarding light regulation<br>Added a warning: `MSG_STATUS_LUMINAIRE_INCOMPATIBEL_SCENE 0xe3`<br>Changed Note: Stopping a sequence loop is possible via a proper reactionTab entry only<br>GetSensorData call now with SystemKey |
| 19.10.2018, Ni<br>23.10.2018, Ni<br><br><br><br>24.10.2018, Ni | 41 | Appendix B<br><br><br><br>Appendix I<br>Appendix E | Description for SetBrightnessRef: Start light regulation<br>GetNodeStatus Byte4, added Bit2: light regulation Tasks active / inactive<br>GetNodeStatus Byte4, added Bit3: movement Task active / inactive<br>SetBrightnessRef: Para 0xffff turns off light regulation<br>Added a warning `MSG_STATUS_MISSING_CONDITION 0xe4`<br>Added correct description for ET_LIGHT_LEVEL, ET_REGULATION |
| 06.11.2018, Ze<br>07.11.2018, Ni | 42 | Appendix Z<br>Appendix C<br><br>Appendix V | Added UART Test<br>Reduced Parameterlist from 5 to 4 Bytes. Changed for AT_LIGHT_CONTROL, AT_EVENT_MANIPULATION, AT_DIM_UP_DOWN_COLOR<br>moveSenseAct: rejected “Auto” and “Auto and On” |
| 12.11.2018, Ni<br><br><br><br><br><br><br><br>11.12.2018, Ni | 43 | Appendix B<br><br><br><br>Appendix C<br><br><br><br>Appendix R<br>Appendix E<br><br><br><br>Appendix K<br>Appendix P | GetNodeStatus, Byte1, Bit4 declares a valid actual timestamp<br>SetTimeStampField renamed in **SetLocalTime**, data timestamp added<br>Added CMD_CHECK_FW_COMPATIBILITY<br>GetTimestamp renamed in **GetLocalTime**<br>Parameter AT_LIGHT_SCENE “Event on target” rejected<br>Changed AT_SEQUENCE_PROCESS para2<br>SetTimer repetition monthly, yearly not required any more<br>Changed due to DT8, added ET_SEQUENCE<br>Description for repetitive TimerEvents reduced for monthly, yearly<br>ET_SYSTEM_TIME_INFO has been rejected and replaced with ET_TIMER in eventTab[]<br>Added ET_SEQUENCE<br>Changed description for sequence end behavior |
| 12.12.2018, Ni<br>20.12.2018, Ni<br><br>02.01.2019, Ni | 44 | Appendix B<br><br><br>Appendix R | Changed parameter CMD_CHECK_FW_COMPATIBILITY<br>Command SetSeqEntry, description byte 0<br>Modified AT_SEQUENCE_PROCESS para1<br>Timer Handling description |

| | | | |
|---|---|---|---|
| 16.01.2019, Ni | | Appendix I | MSG_STATUS_PREMIUM_FAILED rejected |
| 22.01.2019, Ni | 45 | Appendix J<br>Appendix B<br>Appendix C<br>Appendix J | Changed #defines according TechSpec42 for keeping MWAY data model<br>Command SetSeqEntry Byte 3 has been re-changed according TechSpec42<br>AT_SEQUENCE_PROCESS params has been re-changed according TechSpec42<br>Added description details |
| 15.02.2019, Ni | 46 | Appendix C<br><br><br><br><br>Appendix X<br><br>Appendix B<br><br>Appendix E<br>Appendix I: | Changed Note(1): **A reaction of ET_SLIDER with AT_SLIDER_CONTROL acts on a FunctionalGroup which owns a scene number itself. A scene number is therefore not necessary in the App for now, but needs to delivered for message length. A scene number may be used in the future in conjunction with ET_SLIDER_CONTROL i.g. configuration of a scene…**<br>Add slider type spreadsheet for possible combinations of sliderType, ET_SLIDER and ET_SLIDER_CONTROL<br>Added new commands "ConfigMoveSendInterval", "ControlWriteFlash", "ControlEventSend", "IgnoreSetReactionConstraints"<br>ET_SENSE_MOVEMENT: changed parameter Byte 1<br>Added MSG_STATUS_IGNORE_SET_REACTION_CONSTRAINTS (0xe0) |
| 04.03.2019, Ni | 47 | Appendix B | Rejected "ControlWriteFlash", "ControlEventSend" after Telco 19.02.2019 |
| 19.03.2019, Ni | 48 | Appendix B<br><br>Appendix T<br>Appendix J<br>Appendix I | Rejected commands *GetNodeLimitID, SetLimitKey, SetHiddenKey*<br>Cmd SetBrightnessRef para max. value changed<br>Changed description of functional group lightlevel behavior after reset<br>Changed parameter<br>Added MSG_STATUS_INCOMPATIBEL_VIRT_LUM (0x33) |
| 03.04.2019, Ni | 49 | Appendix I | Added MSG_STATUS_GROUP_ADDR_REJECTION_FAILED (0x20) |
| 10.05.2019, Ni | 50 | Appendix D<br>Appendix E<br>Appendix B | Added FirmwareID=1 for NodeType CONNECT_PLUS<br>Added additional ET_REGULATION para<br>GetNodeStatus, Byte 11: represents count of DALI groups<br>SetBrightnessRef new selection for user ref value<br>New command GetNodeVirtLumLL(48) |
| 13.05.2019, Ni | 51 | Appendix B<br><br><br><br><br>Appendix I | Adapted command GetNodeVirtLumLL(48) to GetNodeVirtualLums2(48)<br>New command "GetBrightnessSensorRef"(49)<br>New command "GetNodePhysLum2"(50) with reduced rsp length<br>New command "SetTabledId (51)<br>New command "GetTabledId (52)<br>New MSG_STATUS_LOCAL_TIME_INVALID(0x44), MSG_STATUS_LR_ACTIVE(0xe5), MSG_STATUS_RSP_DATA_OUT_LENGTH_EXCEEDS((0x45) |

| | | | |
|---|---|---|---|
| | | Appendix E | ET_SLIDER with additional control parameter for initial light regulation |

# 2. Abbreviations used in this document

| | |
|---|---|
| BT | Bluetooth |
| BR | Bluerange |
| ET_ | EventType |
| AT_ | ActionType |
| RSP_ | Response |
| Cmd | Command |
| App | Application |
| Virt | Virtual |
| Lum | Luminaire |
| MSB | Most significant bit |
| LSB | Least significant bit |
| Phys | Physical |

# 3. Requirements on BlueRange Mesh

Main topics on the mesh network described in the following chapters of this document:

- The mesh firmware does the complete EnOcean BT button handling on a node (PTM215B, max. 4 units/16 buttons per mesh node) and signals it to VS Firmware via UART incl. Button Key handling etc....
- FW Upgrade via Mesh for the Bluetooth module transferred via GATT and UART (target addressed or complete mesh network).
- FW Upgrade via Mesh for VS µC transferred via GATT and UART (target addressed or complete mesh network).
- The mesh transfers Data according Spec. Appendix A: BT communication frame description
- Implementing of the GATT Protocol. *(will be handled by M-Way)*
- Adjustment of the mesh parameter of the mesh network for light technology requirement (M-Way will optimize Hop Times <20ms, means < 1.28 s for 64 in row connected nodes).
- Mesh network connections should be optimized for short reaction times. *(will be handled by M-Way)*
- Transfer von GATT Nachrichten an UART. *(will be handled by M-Way)*
- The Power down and restart behavior will be optimized by M-Way. The NodeID's will be stored and maintained.
- Offline commissioning with QR-Code of the mesh network. *(will be handled by M-Way)*
- Possibility to add new nodes to an existing mesh network through mesh network. (The GATT connection to existing node has to be used).
- Access to GATT with password in QR-Code.
- Mesh network should handle a timestamp. *(will be handled by M-Way)*

- Auto-Packet Splitting & Reassembling. *(will be handled by M-Way)*
- Possibility to adjust the mesh parameter ‚Count of simultaneous connection in mesh‘ (Standard 4, up to 7) .
- M-Way supplies a simulator which fits to the mesh firmware.
- Beaconing should be able to activate / deactivate via UART/GATT command.
- Commissioning of the nodes via the mesh.
- The mesh should use an unique, unchangeable Mesh-ID. *(will be handled by M-Way)*
- The mesh network should handle up to 400 nodes.
- Source Code and Build-Environment should be available for VS.
- The mesh parameter should be able to request via APP.
- The Node should use FiFo handling.
- The Firmware has to work and run on a module with RF52832 chipset (MBN52832).
- Both parties agree to aim a common SIG certification. Both parties have to support the SIG certification process and necessary adoptions if relevant.
- The ‘Direct Test Mode’ of the **nRF52** must be able to activate via UART command (For Test purposes and certifications). If necessary (e.g. for approvals) a separate firmware file with full HCI support should be available.
- The Source-Code of the Mesh-Firmware and also the App must be available for VS.

- 

## 4. Introduction

**This Document describes the communication between the BlueRange APP and the connected BlueRange Node. It is based on the BlueRange APP description from Mr. Lade.**

***This Document is intended to provide an overview of the needed sequences and commands planned to be used inside the VS-Event-Mesh.***

***This Document is now on a working stage as part of the contract and will only be changed if technical failure is detected and require a change.***

# 5. Definitions

| | |
|---|---|
| **Node** | **A Node is our BT device, per example the Multisensor XS.** |
| **SYSTEM** | **A set of Nodes which will work together (Mesh network).** |
| **NODE KEY** | **This 16 Byte key is used for first configuration of a node, created in VS Production and delivered in the Node QR code.** |
| **Node QR Code** | **QR code with data for first configuration of the Node. It includes Node Key, serial number of the Node, device name etc.** |
| **SYSTEM KEY** | **This Key (16 Byte) is used for encrypting all system and admin communication in the mesh network. The SYSTEM Key will be created by the App at SYSTEM creation.** |
| **USER KEY** | **This Key (16 Byte) is used for encryption of the user communication between the node and connected mobile device (APP). This Key will be created by the APP if a user for the SYSTEM will be created.** |
| **User QR Code** | **A QR code which includes all necessary data for importing a user to the APP.** |
| **Physical Luminaire** | **'real' luminaire connected to the node (ex. DALI can be up to 64 luminaires)** |
| **System / Virtual Luminaire** | **A virtual luminaire is software interface to the real luminaire.** |
| **Encryption** | **We use the XTEA algorithm with different keys (SYSTEM, NODE or USER key) to encrypt our communication.** |
| **Update Level** | **is an up-counting number (0-65535) and will be used to handle the data synchronization between the mobile app(s) and the node(s). Each Node has its own update level. All Commands which are changes the configuration of the node will change the update level of this node. Before the app starts a configuration session, he has to read the update level of the node and compare it with the level stored in the mobile. If the update level in node is higher than the value stored in app, the app has to read the configuration again from the node.** |
| **BRnode software** | **The BRnode software module (M-Way) handles all Bluetooth communications (Mesh, BT switches and GATT connection to mobile devices) and transfers the BRmesh protocol msg's, within the BRnode hardware, to the BTnode software module via UART.** |

| | |
|---|---|
| **BTnode software** | **The BTnode software module (VS) receives / sends the msg's via UART and manages the inputs (sensors) and luminaires within the BRnode hardware.** |
| **Scene** | **is a lightlevel scene (LLScene). A virtual luminaire (or a system luminaire) can output a LLScene thru its channels** |

# 6. Descriptions of the System configuration

All commands are described in Appendix B

## 6.1 Page 'Manage Devices'

**Manage Devices**

- **Add new BR device**: opens QR code scanner (scan 1 to all) cross check and mesh build works in the background (only for selected System / Mesh network)
- **Add Philips Hue device**
- **Add "name"** or location" of new BT device as a must!
- **Delete Devices** (from shown list with radio signal if connected) multiple possible, are you sure? (generate new sys key)

> back Home

### 6.1.1 'Add new BR device'

Scanning Node QR Code.
Build GATT connection to BlueRange device according Info from Node QR Code.
Send command *'SetNodeSystemKey'* with Parameter 'System Key' to add BR device to SYSTEM. Parameter SystemKey "0" resets all node data (scenes, reaction entries, sequence entries, etc.)
Command should be encrypted with Node Key, which is included in Node QR Code.

Send *'GetNodeStatus'* to get all Info about the Node.

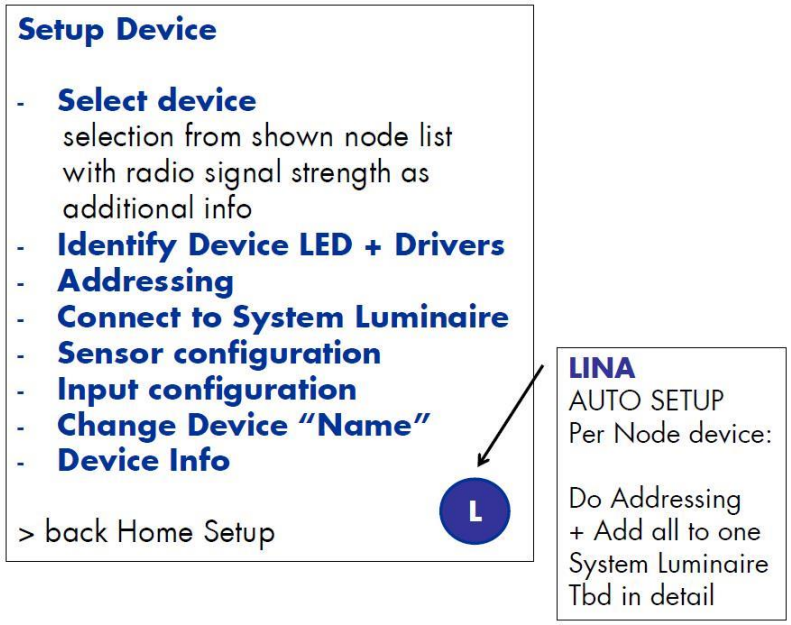
Send *'NodeSearchPhysLum'* to start a search for all, to the BR device connected, physically luminaries (DALI drivers)
Check periodically the state of the node by sending *'GetNodeStatus'* to detect if the search is finished or alternatively wait for the Event 'End of Phys Search'.

Send *'GetNodePhysLums'* to get all data from Node about found physical luminaries.

Note: The QR Code must at least contain the following information (the way of coding them into the QR Code has to be defined as soon as the content is fixed): Node Key, all Codes and information to establish a GATT connection, all information to connect to M-Way mesh.

### 6.1.2 'Add Philips Hue device'

*Not implemented yet. Maybe will be implemented in a future version.*




### 6.1.3 'Delete Devices'

Show the list of all devices . To delete the selected BR device from SYSTEM, **all** other nodes in the SYSTEM have to receive a new SYSTEM KEY with Cmd *'SetNodeSystemKey'* with their, in the SYSTEM stored, NODE KEY.

## 6.2 Page 'Setup Device'

### 6.2.1 'Select device'

Shows a list all available devices (nodes) with radio signal strength.
A device should be selectable for further configuration.

### 6.2.2 'Identify Device + Driver'

Let all connected physical luminaires blink in x seconds interval to identify the device.
Send command *'NodeIdentifyPhysLum'* with the Parameter
Byte 0-1 = #HFFFF (broadcast = all luminaires) and Byte 2 (1 = Identify On / 0 = Identify Off) to switch Identify On or Off.

### 6.2.3 'Addressing'

Start physical Addressing of all physical Luminaires with Cmd '*NodeSearchPhysLum*' .
There are 3 Modes which must be selectable:

Search Mode: 0 (Clear & Search) clear existing addresses and all gets new addresses
1 (Search only New) Only units without short address will get a new short address
2 (Keep & Search) same as Mode 1, but with additional check for duplicity of short addresses

Search Mode is a Parameter of the command above.
Wait while Addressing is running.
Check if finished by periodically checking the Node status with Cmd *'GetNodeStatus'* or by waiting on Event 'End of Phys Search'.
Show all found devices in a list.

### 6.2.4 'Connect to System Luminaire'

1. Make connections between Physical Luminaires and System Luminaire.
2. Create the System Luminaire with Cmd '*NodeCreateVirtualLum*' to get virtual luminaire number.
3. Depending on the wanted Luminaire Type (ex. Tuneable White Luminaire, RGB Luminaire, etc.) the APP sets the necessary count of channels' for the System Luminaire. Assign a System Luminaire channel to a Physical Luminaire address/number with Cmd *'SetPhysLumChannel'*.

General the admin can

- Set new settings of the System Luminaire with Cmd *'SetNodeVirtualLumData'*.
- Read the data of all created System Luminaire by Cmd '*GetNodeVirtualLums*'.
- Read the configured channels of the one System Luminaire num by Cmd '*GetVirtualLumChannels*'.
- Read the Settings of one System Luminaire num with Cmd '*GetNodeVirtualLumData*'.

### 6.2.5 'Sensor configuration'

See Sensor configuration page

#### 6.2.6 'Input configuration'

This changes the input configuration of the selected device. See here.

#### 6.2.7 'Change Device "Name"'

This changes the virtual name of the selected device.

#### 6.2.8 'Device Info'

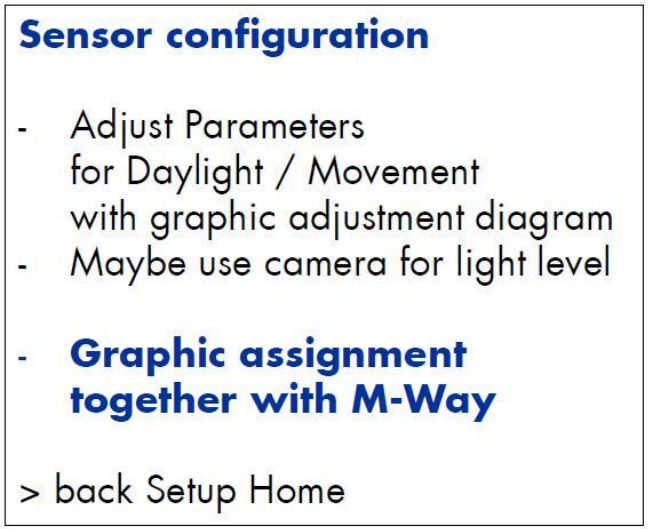
This shows all Information that's available about the selected device.

#### 6.2.9 'LINA AUTO SETUP'

The 'LINA Auto Setup' sets an automatic standard configuration on the selected device.
It starts the following configuration steps:

- Addressing all physical Luminaires with Cmd *'NodeSearchPhysLum'* with Search Mode 0 (Reset & Search).
- Add all physical Luminaires to System Luminaire with procedure shown here.

## 6.3 Page ‘Sensor configuration’



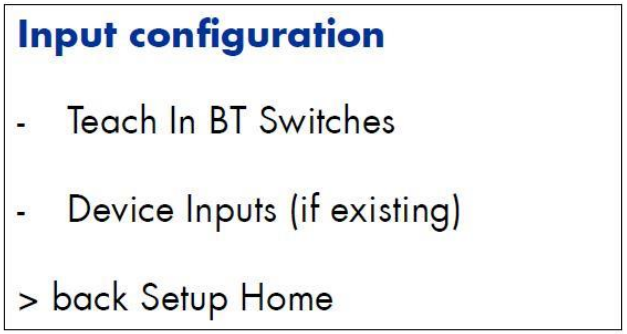

### 6.3.1 Read Sensor Parameters

Read actual Sensor Data by Cmd *‘GetInput’*

### 6.3.2 Write Sensor Parameters

Changed values will be written by Cmd *‘SetInput’*

## 6.4 Page ‘Input configuration’



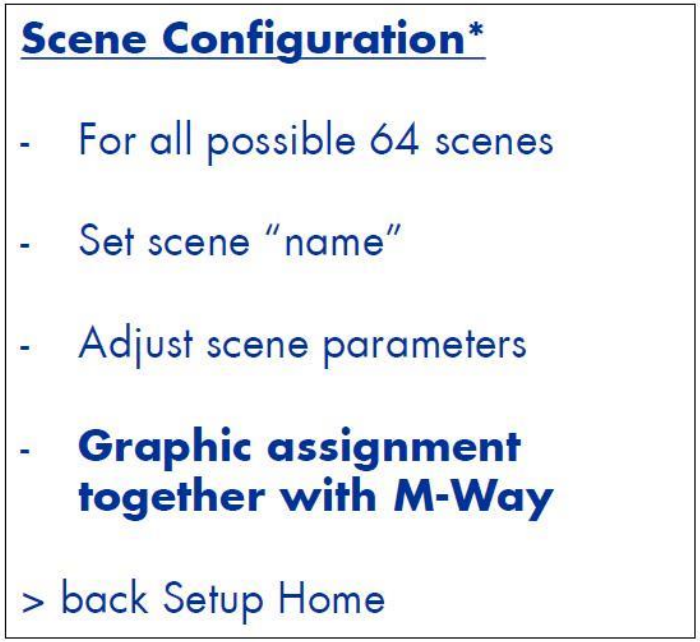

### 6.4.1 Teach In BT Switches

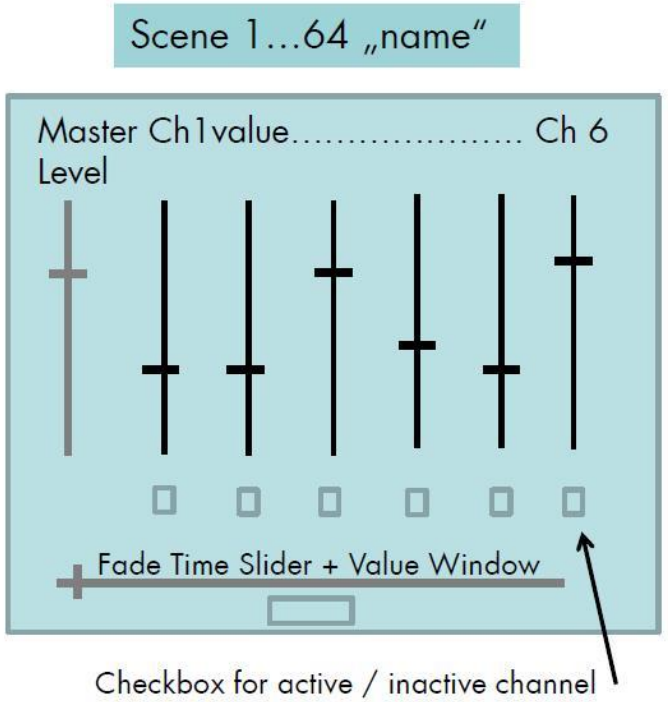
BT Switches have to be taught into the node.
(Paired BT connection with the node)
The Node must create an event which is then being routed into the mesh network.
The Teach In procedure must be approved by VS.

### 6.4.2 Device Inputs

Read data of the Device Input by Cmd *‘GetInput’*.
Sets data of the Device Input by Cmd *’SetInput’.*

## 6.5 Page ‘Scene configuration’




All Level Sliders (vertical) will show a range of 0% to 100%. These percentage values have to be converted into 8 Bit values.
For the Master Level Slider, we are using a linear conversion with following formula:

**x = (100/254) * n** where x = percentage value, n = 8-bit value

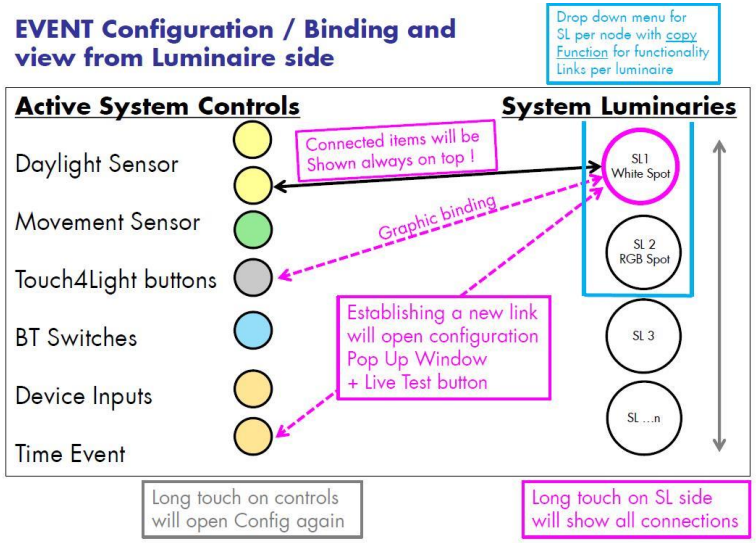
The Channel Sliders (Ch1 … Ch6) using a logarithm conversation as DALI definition:

**x =log10^(((n-1)/(253/3))-1)** (if n = 0, then x must be 0 as well).

The value 255 (0xFF) is reserved for ‘Do not change’.
The Time Slider will show time in hour, minutes or seconds, depending on value. For conversion in a 8 Bit value, we use a range of 32 steps with formula descripted in Appendix Q.

**Scene Parameter can be read by Cmd *‘GetNodeScene’***
**Scene Parameter can be written back to Node by Cmd *‘SetNodeScene’*.**
**To get a live visual feedback about the actual settings, the Cmd ‘SetColorControl’ can be send to the Virtual Luminaire.**

## 6.6 Page ‘EVENT configuration’

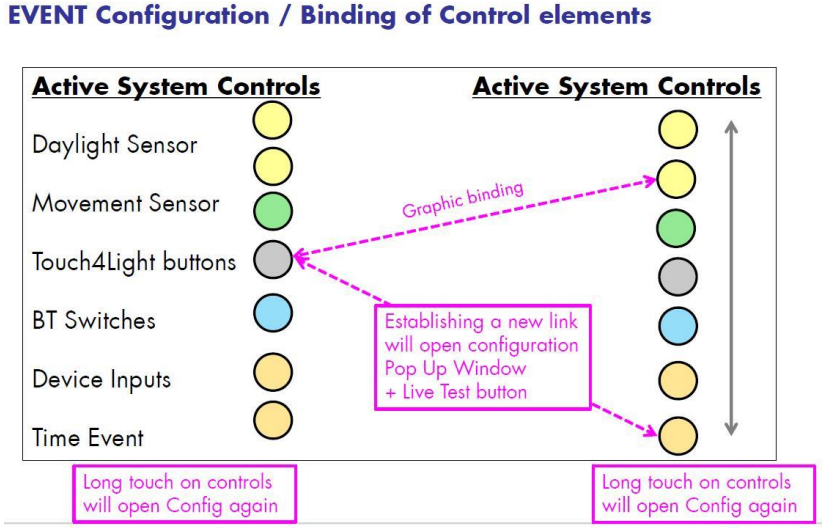



Reaction Configuration is organized as a list with entries.

Actual Event / Input Configuration can read with Cmd *‘GetInput’*.
Actual Reaction Configuration can be read with Cmd *‘GetNodeReactionEntry’*.

Changed Event / Input Configuration can be written with Cmd *‘SetInput’*.

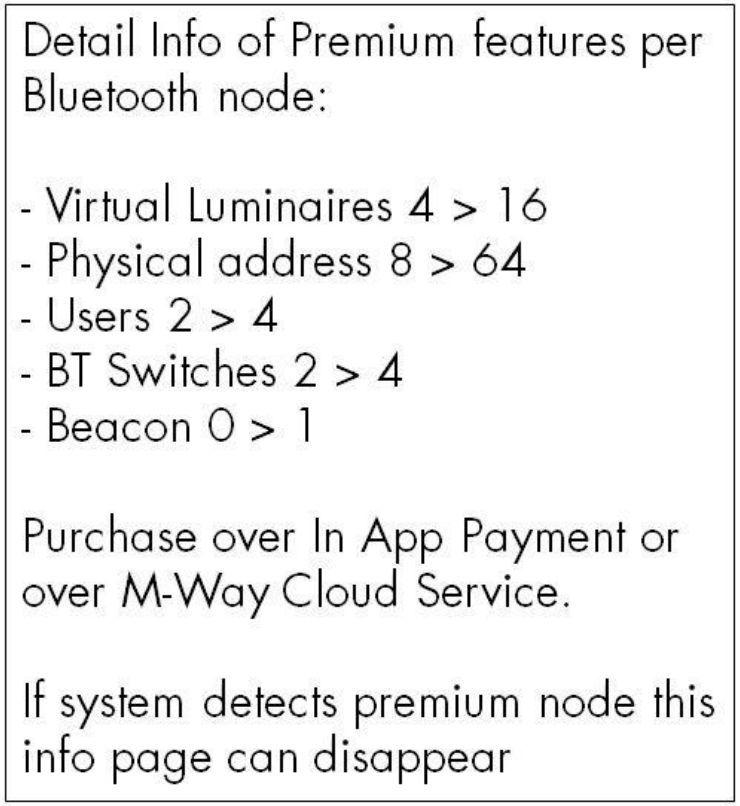
Changed Reaction Configuration can be written with Cmd *‘SetNodeReactionEntry’*.




Manipulation Event Configuration

Manipulation events have to be defined. Probably the same commands described above (with different parameters) can be used.

## 6.7 Page ‘Upgrade Premium Version’

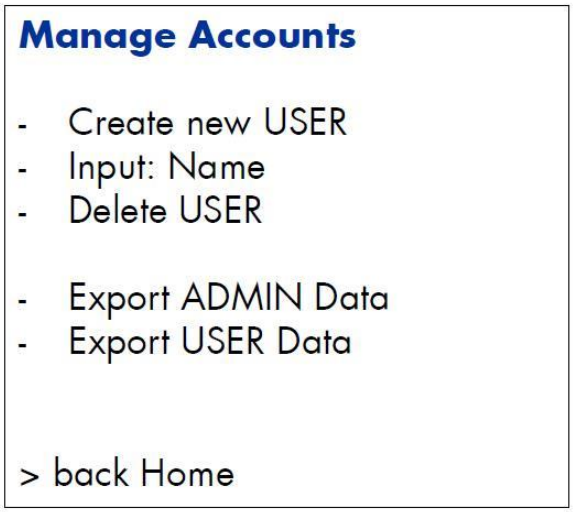



For unlocking the Premium Version after payment, we have defined two commands:

Use Cmd *‘GetNodeLimitId’* to get the Node specific payment data from Node. These data has to be sent to VS to get the Limit Key.

*Use Cmd ‘SetLimitKey’* to unlock premium function in the Node with from payment service generated 128 Bit Key.

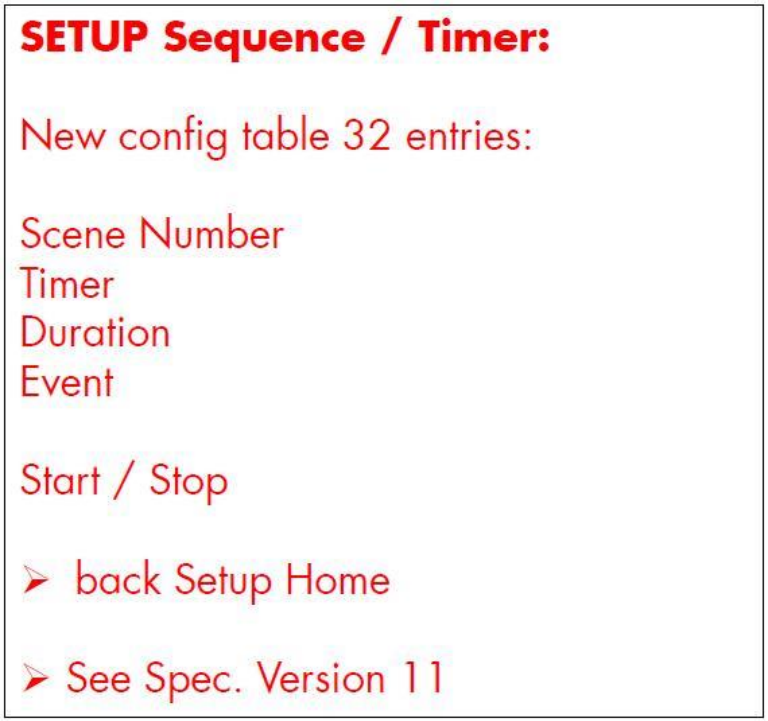
## 6.8 Page ‘Manage Accounts’




### 6.8.1 ‘Create User’

To create a new user, the APP has to create an USER KEY and send it to node with Cmd ‘*SetUser*’.

## 6.9 Page ‘Setup Sequence / Timer:’




**Sequence table entries can be read by Cmd *‘GetSeqEntry’***

**Sequence table entries can be written by Cmd *‘SetSeqEntry’*.**

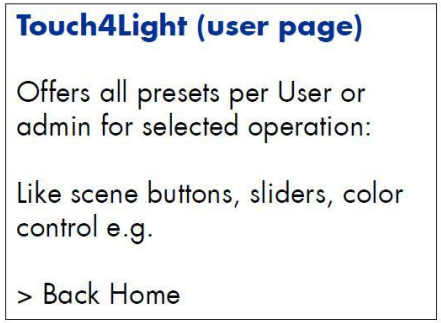
**Timer entries can be read by Cmd *‘GetTimer’*.**
**Timer entries can be written by Cmd *‘SetTimer’*.**

**More info about sequence table, see** Appendix P**.**

**More info about Timers, see** Appendix R**.**

# 7. Descriptions for Touch4Light page (user operating page)

**All touches on soft buttons or sliders on the 'Touch4Light' page, will generate an event (ET_BUTTON, ET_SLIDER, etc.), which will be sent into the mesh network (Packet Type 1).**
**All available Event types are descripted in Appendix E.**




# 8. Appendixes

The appendixes describe in more detail the software design. Generally there are two software parts in a BRnode:

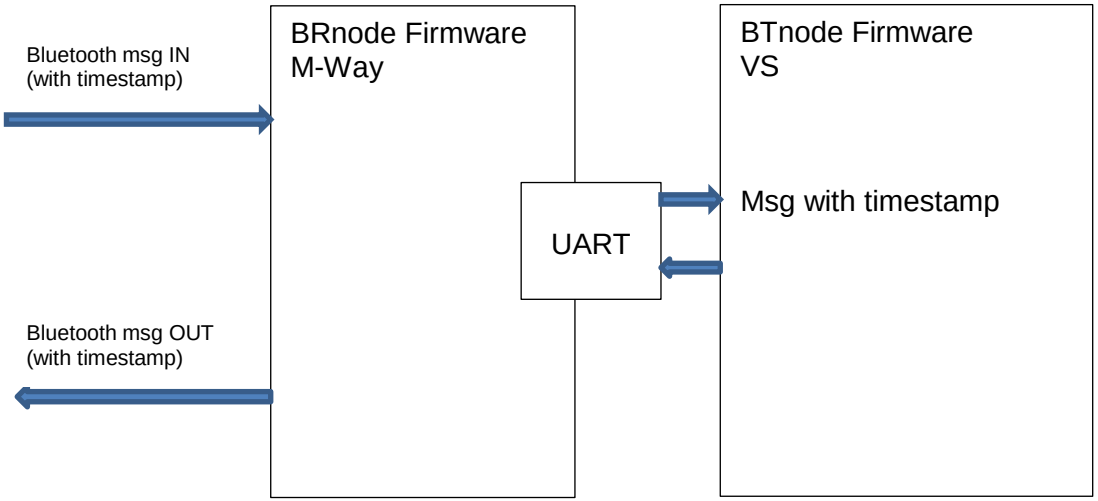
- The **BRnode firmware** (M-Way) handles the in / out of Bluetooth protocol msg's. It transfers / receives messages via UART interface.
- The **BTnode firmware** (VS) receives/transfers UART msg's and manages input events and luminaires.

## 8.1 Appendix A: BT communication frame description

**An incoming message into the BRnode firmware clarifies if the message belongs to the BRnode xyz.**
**This follows that the BTnode firmware receives only messages with its unique DestinationID or broadcast messages.**

| Group | Offset | Size | Field | Value | Description |
|---|---|---|---|---|---|
| - | 0 | 1 | Sync Byte | 0x42 = 66 = ‚B‘ | Serial sync byte; always 0x42 |
| Header | 1 | 1 | Address Length = AL | 0xnn | |
| | 2 | 2 | Block Count = BC | 0xnnnn | Data Length (DL) is BC * 8 – PL – 2 |
| | 4 | 2 | Packet Type | 0xnnnn | See Appendix F |
| | 6 | 1 | Packet Version | 0xnn | |
| | 7 | 2 | Manufacturer | 0xnnnn | 0 = according open standard<br>1 = VS |
| | 9 | 1 | CRC Header | 0xnn | |
| Address | 10 | 1 | Address Status | 0xnn = 0b0f0edcba | Bit field for present Address data<br>(1 = present):<br>a = Source ID field<br>b = Destination ID field<br>c = Destination Type field<br>d = Packet ID field<br>e = Encryption Type field<br>f = Timestamp field |
| | | 2 | Source ID | 0xnnnn | ID of the source node |
| | | 2 | Destination ID | 0xnnnn | ID of destination node |
| | | 1 | Destination Type | 0xnn | 0 = Message between BT-/MWAYfirmware<br>1 = Mesh-Message |
| | | 1 | Packet ID | 0xnn | Necessary to identify response frame |
| | | 1 | Encryption Type | 0xnn | 0 = NodekeyID<br>1 = SystemkeyID<br>N= UserkeyID#N-2; (N=2..251) |
| | | 4 | Timestamp | 0xnnnnnnnn | |
| | | 1 | CRC Address | 0xnn | |
| Data (encrypted) | | 1 | PL | 0xnn | Length of Padding Data |
| | | DL | Data | … | DL = BC * 8 - PL - 2 |
| | | 1 | CRC of Data | 0xnn | CRC of decrypted Data |
| | | PL | Padding data 0x00 | … | Add Padding to reach DL + PL + 2 = BC * 8 |
| | | 1 | CRC Data | 0xnn | |

| Group | Offset | Size | Field | Value | Description |
|---|---|---|---|---|---|
| | | 1 | CRC of CRCs | 0xnn | CRC of CRC Header & CRC Address & CRC Data |



## 8.2 Appendix B: Command description App → MWAYfirmware → VSfirmware

**In General:**

- **If a rxd (uart) message frame fails: it may not possible to select between a command and an event. In this case a simple One-byte-uart response will be delivered with an appropriate MSG_STATUS content.**
- **All commands need to have the correct amount of parameters since the BT-firmware will check the data command length.**
- **All commands will be acknowledged with a response frame (Packet ID 3)**
- **A "scene" in this document is reffered as a "Lightlevel" scene where the user may configure color control of a VLT_TYPE_x**
- **Scene 1 is fallback scene. Will be generated after power on if not exist: brightness: all channels "127", tim2Target=0**
- **All virtLums with phys channels configured will output lighlevel using scene 1 after power up.**
- **Numbers of a virtLum, channel, scene starts with "1" from user perspective. Software intern an array index "0" will be used for "1"**
- **Step%: For example a brightness slider will transmit a value of 0..254, meaning a % brightness between 0..100%. Other parameters with "step%" act in the same way. For example: The user may see a slider setting of 50% in the APP, while the software transfers step% = 127.**
- **Time definitions are based on** (Appendix Q) (0..33).
- **In case of more than 1 Byte parameter transfer: We use BIG-ENDIAN Format: For example a 2-byte-transfer with content 64d results in 0040h .**
- **inputNum: This parameter allows to distinguish between several events of the same type within the reaction tab.**
- **Any settings in any tab requires a free tab entry. Overwriting is possible. In this case the VSfirmware will respond with MSG_STATUS_OVERWRITE_WARNING**
- **The response of a command leads up with the commandNo, following MsgStatus, following 16-Bit updateLevel (MSB first, LSB second)**

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| SetNodeSystemKey | 01 | Set a System Key in a BR Node<br>**Changes Update Level**<br><br>**This command resets all node information and saves the DestinationID in the BTnode firmware** | Byte 0..15: System Key<br>Note: SystemKey “0” will reset complete node data (keys, scenes, virtLums, reactionTab etc.) | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | Node Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| GetNodeStatus | 02 | Request Node Status. | None | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br><br>Byte 4 Status (active=1 / inactive=0):<br>Bit 0: search<br>Bit 1: identify<br>Bit 2: light regulation tasks<br>Bit 3: movement task<br>Bit 4: timestamp valid<br>Byte 5: Node Type (see App. D)<br>Byte 6: Max count physical lums (8 or 64)<br>Byte 7: Count of available physical lums<br>Byte 8: Max count virtual luminaire (4 or 16)<br>Byte 9: Min count virt lum<br>Byte 10: Count …of virt lum<br>Byte 11: … of DALI groups<br>Byte 12: … of virt scenes<br>Byte 13: … of reaction tab entries<br>Byte 14: … of inputs (max TabIndex)<br>Byte 15: … of sequence tab entries<br>Byte 16: …of timer tab entries<br>Byte 17: Max. reactionTab entries<br>Byte 18: Max. User<br><br>**Response data depends also on Node Status 'Free' or 'Premium'.** | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| NodeSearchPhysLum | 03 | Starts the search for physical (DALI-) Luminaries<br>**Changes Update Level** | Search Mode:<br>0 (Clear & Search)<br>1 (Search only New)<br>2 (Keep & Search)<br><br>See also Appendix W | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br><br>Generates a broadcast EventMsg "EndOfPhysicalSearch" | System Key |
| GetNodeUpdateLevel | 04 | Gets the Update Level of the Node | None | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key<br>/<br>User Key |
| GetNodePhysLums (Depreciated, see GetNodePhysLums2) | 05 | Gets the Data of the physical Luminaries | None | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: count of arrays<br>Byte 5..start of Array:<br>Array of Arrays with count of physical Luminaires elements:<br>Pos 0: 2-byte-address Pos 1: 1-byte physical Type<br>(see Appendix J)<br>Pos 2: Status of a *PLT_DALI_LED..:*<br>*Bit0: controlGearFailure*<br>*Bit1: lampFailure*<br>*Bit2: lampOn*<br>*Bit3: limitError*<br>*Bit4: fadeRunning*<br>*Bit5: resetState*<br>*Bit6: shortAddress*<br>*Bit7: powerCycleSeen* | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| NodeIdentifyPhysLum | 06 | Identifies a physical Luminaire | Byte 0/1: Number/address of the physical luminaire: 0x0001..0x0064, 0xFFFE(deviceLED), 0xFFFF (broadcast: all physLums)<br><br>Byte 2: (0 = Identify Off, 1 = On)<br>Note: Process will turn identify OFF automatically after 1 minute ON | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |
| NodeCreateVirtualLum | 07 | Creates a new virtual Luminaire<br>**Changes Update Level** | Byte 0: virtual luminaire number (1..16)<br>Byte 1: VLT Type (Appendix_H) | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |
| GetVirtualLumChannels (maybe obsolete) | 08 | Reads the luminaire channels of a virtual luminaire num | Byte 0: Number of Virtual Luminaire | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Array of 6 Elements<br>Pos 0: Channel number (0xFF = not used) | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| `GetNodeVirtualLums (Depreciated, see GetNodeVirtualLums2)` | 09 | Reads all available data of all virtual luminaires which are available | None | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: count of Array<br>Byte 5..max Byte 66<br>Array of Arrays with count of virtual Luminaires Elements:<br>Pos 0: virtual lum number (1..16)<br>Pos 1: Type, (see Appendix H)<br>Pos 2: Count of channels (which has been set by `SetPhysLum-Channel`)<br>Pos 3: Status | System Key |
| `GetNodeVirtualLumData (maybe obsolete)` | 10 | Reads the settings of a virtual Luminaire<br><br>Note: min brightness is calculated from phys drivers “minLevel”. This value is changeable via App. | Byte 0: Number of Virtual Luminaire | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: dimSpeed<br>Byte 5: modeState<br>(Bit7..4: mode, Bit3..0: state)<br>Byte 6: activeBrightness[step%]<br>Byte 7: passivBrightness[step%]<br>Byte 8: basicBrightness[step%]<br>Byte 9: min brightness[step%]<br>Byte 10: time active<br>Byte 11: time passive<br>(see Appendix Q) (0..33) | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| SetNodeVirtualLumData | 11 | Writes the settings of a virtual Luminaire<br>**Changes Update Level**<br><br>Requires a previous, appropriate NodeCreateVirtualLum SetPhysLumChannel<br><br>See Appendix U | Byte 0: Number Virtual Luminaire(1..16)<br>Byte 1: dimSpeed (0..255) (700ms units)<br>Byte 2: reserved(internal used)<br>Byte 3: reserved(free)<br>Byte 4: activeBrightness[step%]<br>Byte 5: passiveBrightness[step%]<br>Byte 6: basicBrightness[step%]<br>Byte 7: min brightness[step%]<br>Byte 8: time active<br>Byte 9: time passive | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |
| SetPhysLumChannel | 12 | Assigns a Virtual Luminaire channel to a Physical Luminaire address/number<br>**Changes Update Level**<br><br>Requires a previous NodeSearchPhysLum (if not already done so in the past) and an appropriate NodeCreateVirtualLum | Byte 0: Num virt lum(1..16)<br>Byte 1: virt channel num (1..6)<br>Bit7 =1: resets phys channel (0x81...0x86)<br>Byte 2/3: phys channel address/number (DALI:1..64) | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| SetNodeScene | 13 | Sets a scene in Node<br>**Changes Update Level** | Byte 0: Scene Number(1..64)<br>**Note: setting Bit 7 of Byte 0 resets all scene paras (0x81..0xC0)**<br><br>Scene "1" will also be used at PowerOn see Appendix T.<br><br>Byte 1: Brightness (0..254)<br><br>Byte 2: Time to Target (Appendix Q) (0..33)<br><br>Byte 3: Lightlevel channel 1<br>Byte 4: Lightlevel channel 2<br>Byte 5: Lightlevel channel 3<br>Byte 6: Lightlevel channel 4<br>Byte 7: Lightlevel channel 5<br>Byte 8: Lightlevel channel 6<br>(0..254) | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |
| GetNodeScene | 14 | Read a scene from Node | Byte 0: Scene Number(1..64)virtLum2 | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: Room brightness[step%]<br>Byte 5: Time to Target (Appendix Q)<br>Byte 6: Lightlevel channel 1<br>Byte 7: Lightlevel channel 2<br>Byte 8: Lightlevel channel 3<br>Byte 9: Lightlevel channel 4<br>Byte 10: Lightlevel channel 5<br>Byte 11: Lightlevel channel 6 | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| GetNodeReactionEntry | 15 | Reads a Reaction Entry | Byte 0: Number of the Entry (1..250) | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: EventTyp MSB<br>Byte 5: EventTyp LSB<br>Byte 6: SourceID MSB<br>Byte 7: SourceID LSB<br>Byte 8: InputNum<br>Byte 9: ActionTyp<br>Byte 10: UserkeyID (2..251, 255 = all user)<br>Byte 11..17 Para (valid parameter 7 Bytes - depends on ActionType) | System Key |
| SetNodeReactionEntry | 16 | Writes a Reaction Entry<br>**Changes Update Level**<br><br>Depending on ActionType: Requires a previous, appropriate NodeCreateVirtualLum, SetNodeScene | Byte 0: Number of the entry (1..250)<br>Byte 1: EventTyp MSB<br>Byte 2: EventTyp LSB (ET_UNDEF frees tab entry)<br>Byte 3: SourceID MSB<br>Byte 4: SourceID LSB<br>Byte 5: InputNum<br>Byte 6: ActionType<br>Byte 7: UserkeyID (2..251, 255 = all user, 252..254 reserved),<br>Byte 8..xx ,depends on ActionType<br>See also Appendix K | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| GetInput | 17 | Reads complete data of the EventTab | none | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: Entry num x<br>Array of x-Entries<br>Byte 5+x: active (bit oriented)<br>Byte 6+x: enableMesh (bit oriented)<br>Byte 7+x: EventTyp MSB<br>Byte 8+x: EventTyp LSB<br>**Byte 9+x: max Instance(1..8)**<br>Byte 10+x,11+x: see here | System Key |
| SetInput | 18 | Writes the data of the Input<br>**Changes Update Level** | Byte 0/1: EventType<br>Byte 2: active (0,1)<br>Byte 3: enableMesh (0,1)<br>Byte 4: inputNum(0..7)<br>Byte 5,6: Parameter (0xffff will force VSfirmware to set it)<br>See also Appendix E und Appendix K | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |
| SetUser | 19 | Sets a User and its Number. Max.4 users may configure the BRnode.<br>**Changes Update Level** | Byte 0: Position [0.3]<br>Byte 1: UserKeyID[2..251], [255=reset]<br>Byte 2-17: User crypto key | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>: | System Key |
| ~~SetLimitKey~~ | ~~20~~ | ~~Transmit payment 128 Bit Key (This is the key for Payment options)~~ **~~Changes Update Level~~**<br>~~If LimitKey==HiddenKey the premium state will be true~~ | ~~Byte 0: Limit Version (always 0)~~<br>~~Byte 1-16: Key (MSB first)~~ | ~~Byte 0: CmdNo~~<br>~~Byte 1:MsgStatus~~<br>~~(see Appendix I)~~<br>~~Byte 2: updateLevel MSB~~<br>~~Byte 3: updateLevel LSB~~ | ~~Node Key~~ |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| ~~SetHiddenKey (not used any more, since this key will be set during production phase)~~ | ~~21~~ | ~~Transmit hidden 128 Bit Key (This is the key for Payment options)~~ | ~~Byte 0: Limit Version (always 0)~~<br>~~Byte 1-16: Key (MSB first)~~ | ~~Byte 0: CmdNo~~<br>~~Byte 1:MsgStatus (see Appendix I)~~<br>~~Byte 2: updateLevel MSB~~<br>~~Byte 3: updateLevel LSB~~ | ~~Node Key~~ |
| ~~GetNodeLimitId~~ | ~~22~~ | ~~Get ID Code from Node~~ | ~~Byte 0: Limit Version (always 0)~~ | ~~Byte 0: CmdNo~~<br>~~Byte 1:MsgStatus (see Appendix I)~~<br>~~Byte 2: updateLevel MSB~~<br>~~Byte 3: updateLevel LSB~~<br>~~Byte 4-19: DeviceLimitKey (MSB first)~~<br>~~Byte 20-21: Manufacturer ID~~ | ~~Node Key~~ |
| TransmitFirmwareData | 23 | Transmit a new Firmware | Byte 0-3: Firmware Size<br>Byte 4-5: Blocks Total<br>Byte 6-7: Current Block<br>Byte 8-9: Block Size = 256<br>Byte 10-13: Block Start Address<br>Byte 14-xx: encrypted Block Data<br>Byte xx+2:CRC16 of Byte0..xx | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | Node Key |
| CheckFirmwareData | 24 | CRC-Check if the received new VSfirmware is valid | None | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | Node Key |
| SwitchToNewFirmware | 25 | Switch to the new BT firmware | None | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB=0x00<br>Byte 3: updateLevel LSB =0x00 | Node Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| StartFirmwareUpdate | 26 | Starts BT firmware update<br>(this mode follows<br>“TransmitFirmwareData”,<br>“CheckFirmwareData”,<br>SwitchToNewFirmware” ) | none | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | Node Key |
| NodeDeleteVirtualLum | 27 | Deletes a virtual Luminaire (and its physLum assignments)<br>**Changes Update Level** | Byte 0: virtual luminaire number (1..16),<br>255 → deletes all virtLums | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |
| SetLocalTime | 28 | delivers actual Unix UTC local timestamp upon Request Cmd GetLocalTime | Byte 0-4: local UTC time<br>Byte 0 is LSB | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | Node Key |
| SetNodeID | 29 | Sets the node ID | Byte 0-1: DestinationID | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | Node Key |
| GetSeqEntry | 30 | Gets an entry of the sequence table | Byte 0: entry number (1..64) | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: Scene number<br>Byte 5: Pause time<br>Byte 6: Next entry<br>See *SetSeqEntry* | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| SetSeqEntry | 31 | Sets an entry in the sequence table<br>**Changes Update Level**<br><br>**Note:**<br>**A sequence table entry should be free and a scene number must be configured via "SetNodeScene" prior, otherwise warning/error.**<br>See also Appendix P | Byte 0: Entry number [1..64]<br>**Note: setting Bit 7 of Byte 0 resets entry.**<br>Byte 1: Scene number (1..64)<br>Byte 2: Pause time (0..33)<br>Byte 3:<br>Bit 7: Event bit, 1=send<br>Bit 6..0: Next Entry<br>(1..64, 0x7e=stop) | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |
| SetColorControl | 32 | Feedback function:<br>Sets the current light levels and brightness for a virt lum num (for a new BR App slider position) and activates the current light channels (shows the light).<br><br>Requires a previous, appropriate NodeCreateVirtualLum, SetPhysLumChannel | Byte 0: Virt lum num (1..16)<br>Byte 1: brightness (0..254)<br>Byte 2: Lightlevel channel 1<br>Byte 3: Lightlevel channel 2<br>Byte 4: Lightlevel channel 3<br>Byte 5: Lightlevel channel 4<br>Byte 6: Lightlevel channel 5<br>Byte 7: Lightlevel channel 6<br>(0..254) | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key (bei Systemkonfiguration) oder User-Key (bei Touch4Light Oberfläche) |
| GetTimer | 33 | Get a timer in the Node | Byte 0: Timer Number (1..32) | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: Timer Num<br>Byte5-8: timestamp target<br>Byte 9: controlField | System Key |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| SetTimer | 34 | Set a timer in the Node<br>**Changes Update Level** | Byte 0: Timer Number (1..32)<br>Byte 1-4: initial timestamp target<br>Byte 5: Weekday Field<br>(Bit 0=Mon, Bit 1=Tuesday etc.)<br>Byte 6: Control Field<br>(0xFF resets entry)<br>Bit0: daily<br>Bit1: weekly<br>Bit2: ~~monthly~~<br>Bit3: ~~yearly~~<br>(if Bit0-3=0, no repetition, just one event)<br>Bit4: send event to mesh<br>Bit5: enable=1/disable=0 | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | System Key |
| SendRawData | 35 | Send raw data to interface (ex. DALI) for debug purposes (Tester) | Byte 0-7: raw data<br>See Appendix Z | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: driver response data | Node Key |
| GetVirtLumLevel | 36 | Reads the actual brightness and channel levels of a virtual Luminaire. This command maybe helpful for slider sync | Byte 0: Num of Virtual Luminaire | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: brightness[step%][0..254]<br>Byte 5..10: channel1..6 [0..254] | SystemKey / UserKey |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| SetScene2VirtLum | 37 | Sets channel values of a scene to a virtual Luminaire<br>**Changes Update Level**<br><br>Requires a previous, appropriate NodeCreateVirtualLum, SetPhysLumChannel, SetNodeScene | Byte 0: Num of Virtual Luminaire(1..16), (0xFF = all available virtLums)<br>Byte 1: Num of scene(1..64) | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | SystemKey |
| GetSensorData | 38 | Get actual data from sensor (movement & brightness). | none | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: Movement State<br>Or if CONNECT:<br>Bit 1: Button 1 pressed<br>Bit 2: Button 2 pressed<br>Or if CONNECT_PLUS:<br>Bit 1: Button 1 pressed<br>Bit 2: Button 2 pressed<br>Bit 3: Button 3 pressed<br>Bit 4: Button 4 pressed<br>Byte 5..6: Brightness value | SystemKey |
| GetNewSearchData | 39 | Delivers info about newly found phys lum channels after a search (see Appendix W) | none | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4 (MSB) .. Byte 11 (LSB)<br>(Each bit declares phys lum channel whereare Byte 8,Bit0 defines physLum channel address “1” and Byte 1, Bit 7 defines physLum channel address “64” | SystemKey |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| Debug | 40 | Allows to set an Event. Allows to turn on/off Debug outputs<br><br>Note: An "Debug eventType" command will not<br>• Check destinationID,<br>• Check sourceID,<br>• Check user<br>• Send Event to Mesh<br>• Send an EventStatus | Byte 0: type of debug command<br>1: debug output,<br>~~2: set Event~~<br>3: re-program DALI-Groups<br><br>if Byte 0==1 then<br>Parameter Byte 1: 0=off, >0=on<br><br>~~if Byte 0==2 then~~<br>~~Param Byte 1: eventType~~<br>~~Param Byte 2-10: event param1-8~~ | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | NodeKey |
| GetVersion | 41 | Delivers Application-, Bootloader- and Hardware-Version | none | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: Firmware major version<br>Byte 5: Firmware minor version<br>Byte 6: Bootloader major vers.<br>Byte 7: Bootloader minor vers.<br>Byte 8: Hardware major vers.<br>Byte 9: Hardware minor vers. | SystemKey |
| SetBrightnessRef | 42 | Sets a reference control value for light regulation.<br>Stops and starts light regulation depending on parameter | Byte 0: reference MSB<br>Byte 1: reference LSB<br>*Note:*<br>*0xFFFF: stops light regulation*<br>*0xFFFE: takes mean sensor reading*<br>*0..9999: takes user ref value* | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | SystemKey |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| CheckFirmwareCompati-bility | 43 | „updatePackage"-Handling für unterschiedliche nodeDevice Komponenten. Für eine Komponente von FirmwareType „1" werden nur bestimmte nodeDeviceTypen aktualisiert. | Byte 0: 0<br>Byte 1: 0<br>Byte 2: minor version<br>Byte 3: major version<br>Byte 4: FirmwareType / ComponentID | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br><br>*Wenn Response CMD_UNKOWN hat das nodeDevice noch einen älteren Softwarestand und es kann direkt mit „StartFirmwareUpdate" fortgefahren werden.*<br><br>*Wenn Response WRONG_DESTINATION ist das updatePackage nicht für das nodeDevice gedacht. Ein folgendes „StartFirmwareUpdate" wird gleichermaßen abgewiesen.* | Node Key |
| ConfigMoveSendInter-val | 44 | Configures how often a detected movement signal should be send into mesh | Byte 0: 1..255<br>1=1*700ms<br>2=2*700ms<br>..<br>255=255*700ms=ca.3min | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | NodeKey |
| Reserved for future use | 45 | | | | |
| Reserved for future use | 46 | | | | |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| SetNodeReactionEntryNoConstraints | 47 | This is an adaption of SetNodeReactionEntry command, but:<br>Allows reactionTab settings although physDev, scene, timer, sequences etc. may not available by the time SetNodeReactionEntryNoConstraints has been received. If not available a warning MSG_STATUS_IGNORE_SET_REACTION_CONSTRAINTS will be delivered. | Same as *SetNodeReactionEntry* | Same as *SetNodeReactionEntry* | SystemKey |
| GetNodeVirtualLums2 | 48 | Same as GetNodeVirtualLums in addition actual light level. | None | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: count of Array<br>Byte 5..start of array's:<br>Array:<br>Pos 0: virtual lum number(1..16)<br>Pos 1: Type, (see Appendix H)<br>Pos 2: Count of channels (which has been set by SetPhysLumChannel)<br>Pos 3: Status<br>Pos 4: actual light level | SystemKey |
| GetBrightnessSensorRef | 49 | Retrieves the sensor brightnes reference for light regulation (which is enclosed in "eventTab.parameter") | None | Byte 0: CmdNo<br>Byte 1:MsgStatus (see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: ref MSB<br>Byte 5: ref LSB | SystemKey |

| CommandTypMsg | Cmd/Rsp-No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| GetNodePhysLums2 | 50 | Gets the Data of the physical Luminaries with reduced response data | None | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: count of arrays<br>Byte 5..max Byte 197:<br>Array of Arrays with count of physical Luminaires elements:<br>Pos 0: 1-byte-address<br>Pos 1: 1-byte physical Type<br>(see Appendix J) | System Key |
| SetTabletId | 51 | Sets tablet ID for copy / paste APP-configuration. Used by App only | Byte 0: MSB Id<br>Byte 1: LSB Id | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB | NodeKey |
| GetTabletId | 52 | Gets tablet ID for copy / paste APP-configuration. Used by App only | none | Byte 0: CmdNo<br>Byte 1:MsgStatus<br>(see Appendix I)<br>Byte 2: updateLevel MSB<br>Byte 3: updateLevel LSB<br>Byte 4: MSB Id<br>Byte 5: LSB Id | NodeKey |

**A BTresponse upon a BTcmd will use the BT communication frame with Packet ID 3**

## 8.3 Appendix C: Reaction List description

**Incoming BT-Event messages will be handled internally in the BTnode software with a reaction tab. The tab need to be setup via BTcommand "SetNodeReactionEntry" for each desired action. Each action is represented by one tab entry. Max. 250 entries are possible.**

### Structure of one entry in a reaction tab[max.250]

| Name | Description | Byte count |
|---|---|---|
| Event Type (see Appendix E) | The type of an input<br>(Brightness / Movement / Button / Lightlevel / etc.) | 2 |
| Event SourceID | SourceID | 2 |
| Event inputNum | inputNum<br>(each input in the node has its own inputNum | 1 |
| Action Type | Defines the reaction of a BTevent | 1 |
| User number | Defines the accessibility of the reaction<br>(0xFF == free access) | 1 |
| Parameter | depends on Action Type (see AT_Parameter) | 1 to 6 |

### ActionType

#define AT_UNDEF (0)

#define AT_LIGHT_SCENE (1)
Outputs a lightlevel to the specified virtLum type channels. Lightlevel will be retrieved from the scene channels. Scene channels which are defined but not matched with virtLum type will be ignored.
If mode=auto: the incoming event will be used as a "time button" – which means light control is active (this is only possible, if a time active/passive has been configured for the system luminaire). Light regulation is active as long active time has been expired.
If mode=manual: light control and light regulation will be rejected if processed.

#define AT_DIM_UP_DOWN_BRIGHTNESS (2)
Each event outputs a lightlevel: The brightness will change over "stepCount %" while the color schema is fix according the specified scene.

#define AT_DIM_UP_DOWN_COLOR (3)
Based on the mean brightness value of both scenes: Each event outputs a lightlevel of "stepCount lightlevel" for the specified color channels in the *channelSelector byte*. Non specified channels in *channelSelector* outputs a fix mean channel lightlevel of both scene channel level.

#define AT_LIGHT_CONTROL (4)
A virtLum in manual mode forces a lightlevel change by an user action. An automatic mode triggers a lightlevel change by a timer or sensor. The mode change can be configured with e.g. an EV_BT_BUTTON or a “Touch for Light button” on the HandyApp.

#define AT_TIMER (5) //obsolete

#define AT_EVENT_TAB_MANIP (6)
Allows a parameter change in the eventType structure. See Appendix C

#define AT_SEQUENCE_PROCESS (7)
See Appendix P

#define AT_LIGHT_REGULATION (8)
Internal use in VSfirmware: takes action on FG due to an external light regulation

#define AT_SLIDER_CONTROL (9)
If ET_SLIDER_CONTROL: Sets channels of a virtLum directly, if ET_SLIDER a slider type will be used (channel calulations by VSfirmware)

#define AT_BEACON_STATE (10)
An event (e.g. ET_BT_BUTTON_TYPE_x) can switch indirectly Beacon function On/Off/Toggle by sending appropriate message VSfirmware → MWAYfirmware

## Action Types with Parameter Field

| ActionTyp | num Para | Parameter 1 | Param 2 | Param 3 | Param 4 |
|---|---|---|---|---|---|
| AT_LIGHT_SCENE | **4** | Virt. Lum No[1..16] | Scene No[1..64] | Not used | Mode Manual = 0, Auto = 1 |
| AT_DIM_UP_DOWN_BRIGHTNESS | **4** | Virt. Lum No[1..16] | Up[1] / Down[0] | Scene No[1..64] | StepCount [0..255] |
| AT_DIM_UP_DOWN_COLOR | **4** | Virt. Lum No[1..16] | ChannelSelect (bit: d0CC CCCC) Bits "C" Selects channels Bit d=1 DIMM UP Bit d=0 DIMM DOWN | To Scene [1..64] | StepCount [0..255] |

| | | | | | |
|---|---|---|---|---|---|
| AT_LIGHT_CONTROL | **4** | Virt. Lum No[1..16] | changeLL (activeLL, passiveLL, basicLL)<br>Description: changes actualLL to "changeLL" | moveSenseAct (Appendix V) | Scene No[1..64] |
| AT_EVENT_MANIP | **4** | Event Type | sensorActive(low half byte)<br>enableMesh(high half byte)<br>for each half byte:<br>Disable = 0<br>Enable = 1<br>Toggle=2<br>Keep=4 | Value | Value |
| AT_SEQUENCE_PROCESS | **2** | Virt. Lum No[1.16] | startEntry(1..64, 0x7E=stop, 0xFE=stop & enableEvent) | | |
| AT_LIGHT_REGULATION | **2** | Virt. Lum No[1..16] | Scene[1..64] | | |
| AT_SLIDER_CONTROL (1) | **3** | Virt. Lum No[1..16] | slider type (see Appendix X) | Scene[1..64] | |
| AT_BEACON_STATE (2) | **1** | State (Off=0, On=1,Toggle= 2) | | | |

**Notes:**

**(1) A reaction of ET_SLIDER with AT_SLIDER_CONTROL acts on a FunctionalGroup which owns a scene number itself. A scene number is therefore not necessary in the App for now, but needs to delivered for message length. A scene number may be used in the future in conjunction with ET_SLIDER_CONTROL i.g. configuration of a scene…**

**(2) The action AT_BEACON_STATE sends `SetBeaconState,see` Appendix L**

## 8.4 Appendix D: List of Node Types

| NodeType | #define | FirmwareID | HW-Input | Sensor | Dali | Comment |
|---|---|---|---|---|---|---|
| UNDEF | 0 | - | - | - | - | |
| MULTISENSOR_XS | 1 | 1 | 0 | Movement, Brightness | yes | |
| CONNECT | 2 | 1 | 2 | - | - | Connection of 2 HW-switches |
| MULTISENSOR_XL | 3 | 1 | 2 | Movement, Brightness | yes | sensor range bigger |
| CONNECT_ME | 4 | 1 | 0 | - | yes | |
| ~~E27~~ | ~~5~~ | ~~2~~ | ~~0~~ | ~~-~~ | ~~-~~ | |
| ~~E27_RING~~ | ~~6~~ | ~~2~~ | ~~0~~ | ~~-~~ | ~~-~~ | |
| DIGI_LED_BLU | 7 | tbd | 0 | - | yes | 4-channel-driver with bluetooth |
| CONNECT_PB4 | 8 | 1 | 4 | - | - | Connection of 4 HW-switches |
| | | | | | | |
| | | | | | | |

## 8.5 Appendix E: Event (Input) Types

**//possible Node events for all type of known deviceNodes**

```
#define ET_UNDEF                (0)     //undefined
#define ET_BT_BUTTON_TYPE_1     (1)     //a bluetooth button with four wipes to be learned in by App
#define ET_BT_BUTTON_TYPE_2     (2)     //a bluetooth button with four wipes to be learned in by App
#define ET_BT_BUTTON_TYPE_3     (3)     //a bluetooth button with four wipes to be learned in by App
#define ET_BT_BUTTON_TYPE_4     (4)     //a bluetooth button with four wipes to be learned in by App
#define ET_BUTTON               (5)     //AppEvent, push button
#define ET_SENSE_MOVEMENT       (6)     //movement sensor
#define ET_SENSE_BRIGHTNESS     (7)     //brightness sensor
#define ET_LIGHT_LEVEL          (8)     //used for reactionTab to define FunctionalGroups which belongs to light regulation
#define ET_SLIDER               (9)     //AppEvent, sets channel(s) according slider position
#define ET_TIMER                (10)    //internal VS event to start time controlled action
#define ET_REGULATION           (11)    //event to adjust light output for an external light regulation
#define ET_SLIDER_CONTROL       (12)    //AppEvent, event reflects SetColorControl
#define ET_EXT_BEACON           (13)
#define ET_IO_BUTTON            (14)    //for NT_VS_CONNECT
#define ET_SEQUENCE             (15)    //internal VS event to start an action upon “end of a sequence”
```

```
//eventType's used from BT to BR firmware
#define ET_CHANGE_SEQUENCE      (253)
#define ET_POWER_READY          (254)
#define ET_END_OF_PHYS_SEARCH   (255)
```

To react on its own input events, an appropriate entry in the reaction tab is needed via the “Bluerange” BR App

## Structure of one entry in an Event Tab[num of inputs]

| Name | Description | Byte count |
|---|---|---|
| inputActive | An inputNum0..7 can be bitwise(0..7) enabled=1 / disabled=0<br>configurable via AT_EVENT_MANIP BTevent msg | 1<br>(default=0) |
| enableMesh | Once inputActive and and enableMesh are enabled, an event registration can be broadcast thru the BT mesh network<br>configurable via AT_EVENT_MANIP BTevent msg | 1<br>(default=0) |
| eventType | Button or Brightness or Movement or Lightlevel or… | 2 |
| Parameter | Reserved | 2 |

The event tab length is fixed for a BRnode (node) type and reflects the “Count of available Inputs” of a BRnode.

- **Following EventMsgs with PacketType=PT_EVENT exist.**
- **Encryption type is UserKey if send from HandyApp, otherwise SystemKey**
- **All EventMsgs will be acknowledged just with one UART byte directly**. It will be delivered after the availability of settings has been checked but before the execution of the action will be fulfilled.
- **Sending Events from HandyApp m u s t use an (user) encryption type >= 2 in address field of the BT msg. This implements also a correct user number** (0xFF == all users) **in the reactionTab.**
- **Sending Events not from HandyApp m u s t use system encryption type = 1 in address field of the BT msg. This implements also a correct user number = 0xff in the reactionTab.**

| EventMsg | Event No. | Description | Parameter | UART-Response |
|---|---|---|---|---|
| Button | ET_BUTTON | A smartphone / tablet Button-Event. Sends a button press / hold / release<br>If “press”→ event time is 0<br>If “hold” → event time is xx and defines a dimming request<br>If “release” → event time is xx | Byte 0: input-Num(0..255)<br>Byte 1: press(1), release(2), hold(3) | Byte 0: MsgStatus |

| **EventMsg** | **Event No.** | **Description** | **Parameter** | UART-**Response** |
|---|---|---|---|---|
| `ButtonBT` | ET_BT_BUTTON_TYPE[1..4] | A Bluetooth-Button Event. Sends a button press / hold / release<br>If "press"→ event time is 0<br>If "hold" → event time is xx and defines a dimming request<br>If "release" → event time is xx | Byte 0: inputNum(0..4)<br>Byte 1: press(1), release(2), hold(3) | Byte 0: MsgStatus |
| `Slider` | ET_SLIDER | A smartphone / tablet Slider-Event. Sends a new slider position to a slider type | Byte 0: inputNum<br>Byte 1: position<br>Byte 2: control:<br>0=normal,<br>1=LR Stellglied | Byte 0: MsgStatus |
| `SliderControl` | ET_SLIDER_CONTROL | A smartphone / tablet Slider-Event without LLscene processing. Sends a new channel setting to a virtLum | Byte 0: inputNum<br>Byte 1: brightness (0..254)<br>Byte 2: channel 1(R)<br>Byte 3: channel 2(G)<br>Byte 4: channel 3(B)<br>Byte 5: channel 4(W)<br>Byte 6: channel 5(A)<br>Byte 7: channel 6(F) (0..254) | Byte 0: MsgStatus |
| `SystemTime` (unused for now) | ET_SYSTEM_TIME | Sends the BRnode node system time (unused for now) | Byte 0: inputNum<br>Byte 1-4: sysTime (MSB first) | Byte 0: MsgStatus |
| `Movement` | ET_SENSE_MOVEMENT | Sends a movement sensor status | Byte 0: inputNum<br>Byte 1:<br>1=new Event if timer interval elapsed<br>1+C="C" counts detected within timer interval<br>Timer interval: default = 11, which is 11*770ms<br>See also `Config-MoveSendInterval` | Byte 0: MsgStatus |

| **EventMsg** | **Event No.** | **Description** | **Parameter** | UART-**Response** |
|---|---|---|---|---|
| Brightness | ET_SENSE_BRIGHTNESS | Sends a brightness sensor info (every 8 sec) | Byte 0: inputNum<br>Byte 1/2: current-BrighnessValue<br>Byte 3/4: targetBright-nessValue | Byte 0: MsgStatus |
| *LightLevel* | *ET_LIGHT_LEVEL* | *VSfirmware only: Defines which virtLums belongs to own active light regulation within reaction tab* | | |
| Timer | ET_TIMER | Event to trigger a time controlled action | Byte 0: TimerNum(1.32) | Byte 0: MsgStatus |
| *Regulation* | *ET_REGULATION* | *VSfirmware only: Defines which virtLums belongs to external light regulation within reaction tab. Receiving this event forces those virtLums to change its light level via DAPC independent of byte1.*<br>*Additional control is possible via "control" byte1.* | *Byte 0: DAPC [0..254]*<br>*Byte 1: control[0,1]:*<br>*0= set DAPC only*<br>*1= init Auto active* | *Byte 0: MsgStatus* |
| ExtBeacon | ET_EXT_BEACON | This event will be sent by BR firmware if an external matching Beacon is recognised | Byte 0: inputNum( == Beacon ID) | |
| IO_Button | ET_IO_BUTTON | The I/O-Inputs of a Connect device sends this event for button press / hold / release.<br>If "press"→ event time is 0<br>If "hold" → event time is xx and defines a dimming request<br>If "release" → event time is xx | Byte 0: inputNum(0..1)<br>0 = Input 1, 1 = Input 2<br>Byte 1: press(1), re-lease(2), hold(3) | Byte 0: MsgStatus |
| | | | | |
| *PowerReady* | *ET_POWER_READY* | *a)This Event can be sent by the MWAYfirm-ware as request.*<br>*b)Send by VSfirmware as soon the µC voltage has been stabilized or as a response upon a request.*<br>*After receiving this Event by the MWAYfirm-ware, it can start its Bluetooth process.* | *Byte 0: (1 ready, 0 not ready)* | |
| *EndOfSequence* | *ET_ SEQUENCE* | *Generated by VS at the end of a sequence, if reactionTab set accordingly.* | *Byte 0: inputNum = SeqNum(tbd)* | |

| EventMsg | Event No. | Description | Parameter | UART-Response |
|---|---|---|---|---|
| *EndOfPhysi-calSearch* | *ET_END_OF_PHYS_SEARCH* | *Upon a BTCmd "NodeSearchPhysLum" this BTEvent msg will send by BT firmware if the search of physical luminaire ends up.* | *Byte 0-8: all available physical luminaire addresses, MSB first. Setting/Resetting of Bit63..Bit0 indicates address 64..1 found(=1) / not found(=0)* | |

The following table lists available EventType for **NT_VS_MULTISENSOR_EM_E** and can be used to set reactionTab appropriate regarding sourceID, inputNum and user:

| | | | | BTfirmware check within ReactionTab | | | | |
|---|---|---|---|---|---|---|---|---|
| EcentType | enclosed in eventTab for NT_VS_MULTISENSOR_EM_E | intern source events | extern Events (not from extern node) | sourceID | inputNum | user | AppEvent | ManualEvent |
| ET_BT_BUTTON_TYPE_1 | x | | x | y | y | n | | x |
| ET_BT_BUTTON_TYPE_2 | x | | x | y | y | n | | x |
| ET_BT_BUTTON_TYPE_3 | x | | x | y | y | n | | x |
| ET_BT_BUTTON_TYPE_4 | x | | x | y | y | n | | x |
| ET_SENSE_MOVEMENT | x | x | | y | n | n | | |
| ET_SENSE_BRIGHTNESS | x | x | | y | n | n | | |
| ET_LIGHT_LEVEL | x | x | | y | n | n | | |
| ET_SYSTEM_TIME_INFO | x | | | y | n | n | | |
| ET_REGULATION | x | x | | y | n | n | | |
| ET_EXT_BEACON | x | | x | y | y | n | | |
| ET_BUTTON | | | x | n | y | y | x | x |
| ET_SLIDER | | | x | n | y | y | x | x |
| ET_SLIDER_CONTROL | | | x | n | y | y | x | x |

#define ET_IO_BUTTON 30 is used for node type NT_BLUERANGE_CONNECT

Special Events, not listed in EventTab
*VSfirmware → MWAYfirmware:*
#define ET_CHANGE_SCENE 0x00FD //as a notice for "nextEntry" (or "finished") in scene sequence
#define ET_END_OF_PHYS_SEARCH 0x00FF

*VSfirmware ←→ MWAYfirmware:*
#define ET_POWER_READY 0x00FE //as request from MWAYfirmware / as a power ready state from VSfirmware

## 8.6 Appendix F: List of Packet- and Destination-Types in MsgHeader

| PacketType | | description |
|---|---|---|
| PT_UNDEF | 0 | undefined |
| PT_EVENT | 1 | Event |
| PT_CMD | 2 | Command |
| PT_RSP | 3 | Response |

The PacketType defines the overall content in the Data-Block of the communication frame. (Appendix A).
For PacketType 2 and 3, the content is listed in Appendix B, for each command.
For PacketType 1, see Appendix C.

| DestinationType | | description |
|---|---|---|
| DT_TYPE_LOCAL | 0 | Internally generated events which also processed on the same node<br>Received events from node coupled devices (e.g. BT_BUTTON_TYPE_X) |
| DT_TYPE_MESH | 1 | Received commands / events from mesh<br>Sending commands / events into mesh |

## 8.7 Appendix G: Definition of User Dataset (for User 2D Code Preparation)

**The dataset for the User 2D Code includes the following information:**

- Mesh Name
- User number
- User Key
- GATT Key
- Update Level
- List of Nodes which the User can connect to

**Will be extended in cooperation with M-Way, if necessary!**

## 8.8 Appendix H: Definition of the System Luminaire Types

| Virtual LumType | | description |
|---|---|---|
| VLT_AVAILABLE_FOR_THIS_TYPE | 0 | an intern BT firmware value. Some or all channels may be available |
| VLT_SINGLE_LUMINAIRE | 1 | fix 1-channel luminaire (a channel 0 config only, all other will result in an error) |
| VLT_TUNEABLE_WHITE_LUMINAIRE | 2 | fix 2 channel luminaire (a channel 0,1 config only, all other will result in an error) |
| VLT_RGB_LUMINAIRE | 3 | fix 3 channel luminaire (a channel 0,1,2 config only, all other will result in an error) |
| VLT_RGBW_LUMINAIRE | 4 | fix 4 channel luminaire (a channel 0,1,2,3 config only, all other will result in an error) |
| VLT_RGBWA_LUMINAIRE | 5 | fix 5 channel luminaire (a channel 0,1,2,3,4 config only, a config on channel 5 will result in an error) |
| VLT_RGBWAF_LUMINAIRE | 6 | fix 6 channel luminaire (a full channel 0,1,2,3,4,5 configuration required) |
| | | |
| VLT_UNAVAILABLE | 255 | an intern BT firmware value. Some channels may be unavailable |

## 8.9 Appendix I: Msg_Status

**The MsgStatus will be sent in a BTresponse msg from the BTnode software module upon receiving a correct encrypted BTcommand msg from the BRnode software module.**

**A BTresponse or an UART-Response apon an extern Event may have one of the following status codes:**

```
//Errors
#define MSG_STATUS_OK                                0x00 //no problems has been detected
#define MSG_STATUS_WRONG_DATALENGTH                  0x01 //The data length of command / event  dissatisfies
#define MSG_STATUS_NOT_ACCEPTED_DURING_SEARCH        0x02 //a command has been received while searching physical luminaire channels
#define MSG_STATUS_NO_FREE_VIRT_LUM_CHANNEL          0x03 //a virtLum type could not be created
#define MSG_STATUS_UNAVAILABLE_VIRT_LUM_NUM          0x04 //a luminaire number of a command para doesn't exist
                                                          //an reaction func detects missing luminaire number
#define MSG_STATUS_INVALID_PHYS_LUM_ADDR             0x05 //used in CMD_NODE_IDENTIFY_PHYS_LUM
#define MSG_STATUS_PHYS_LUM_NUM_NOT_DETECTED         0x06 //a virtLum channel can be used only if the channel group address exist
#define MSG_STATUS_SCENE_NOT_AVAILABLE               0x07 //the command or the Re-action cannot be placed: missing lightLevelScene num
#define MSG_STATUS_CHANNEL_NOT_FREE                  0x09 //not used anymore / for now
#define MSG_STATUS_REACTION_ENTRY_NOT_AVAILABLE      0x09 //intern SW error of VSfirmware
#define MSG_STATUS_ACTION_TYPE_NOT_AVAILABLE0x0a //the command SetNodeReactionEntry failed
#define MSG_STATUS_WRONG_EVENT_TYPE                  0x0b //the command SetNodeReactionEntry failed
#define MSG_STATUS_TIME_EXCEEDS                      0x0c //wrong parameter in SetNodeScene
#define MSG_STATUS_GROUP_CREATION_FAILED             0x0d //the command SetPhysLumChannel failed
#define MSG_STATUS_CHANNEL_LIMIT_ERROR               0x0e //creating a group address > 16 failed
#define MSG_STATUS_VIRT_LUM_TYPE_CHANNEL_UNAVAILABLE 0x0f //creating a virtLum type failed: not enough group addresses
#define MSG_STATUS_NO_SENSOR_AVAILABLE               0x10 //the type of the nodeDevice does not have any sensor connected
#define MSG_STATUS_WRONG_VIRT_LUM_INDEX              0x11 //A reaction could not be placed
#define MSG_STATUS_WRONG_SCENE_INDEX                 0x12 //A reaction could not be placed
#define MSG_STATUS_WRONG_SEQUENCE_INDEX              0x13 //A reaction could not be placed
#define MSG_STATUS_CMD_REQUIRES_SYSTEM_KEY           0x14 //the command has wrong encryption key
```

```
#define MSG_STATUS_CMD_REQUIRES_NODE_KEY          0x15 //the command has wrong encryption key
#define MSG_STATUS_CMD_UNKOWN                     0x16 //wrong cmdID received

#define MSG_STATUS_RXD_DATA_CRC_ERR               0x17 //the message receiver detects an error
#define MSG_STATUS_RXD_HEADER_CRC_ERR             0x18 //the message receiver detects an error
#define MSG_STATUS_RXD_MAX_LENGTH_ERR             0x19 //the message receiver detects an error
#define MSG_STATUS_RXD_LESS_DATA_ERR              0x1a //the message receiver detects an error
#define MSG_STATUS_RXD_ADDR_CRC_ERR               0x1b //the message receiver detects an error
#define MSG_STATUS_RXD_TOTAL_CRC_ERR              0x1c //the message receiver detects an error
#define MSG_STATUS_RXD_FRAME_ERR                  0x1d //the message receiver detects an error
#define MSG_STATUS_RXD_HEADER_ADDRESS_LENGTH0x1e //the message receiver detects an error
#define MSG_STATUS_RXD_PACKET_ID_ERR              0x1f //the message receiver detects an error
#define MSG_STATUS_GROUP_ADDR_REJECTION_FAILED    0x20 //verification of removing addr within group failed
#define MSG_STATUS_UPDATE_NOT_STARTED             0x21 //TransmitFirwareData failed
#define MSG_STATUS_FIRMWARE_INCOMPLETE            0x22 //CheckFirmwareData failed
#define MSG_STATUS_CHECKSUM_NOT_OK                0x23 //CheckFirmwareData failed
#define MSG_STATUS_USER_NOT_AVAILABLE             0x24 //SetNodeReactionEntry failed or reactionTab has no appropriate user entry
#define MSG_STATUS_EVENT_NOT_ACTIVATED            0x25 //Event not activated in inputEventTab
#define MSG_STATUS_WRONG_DESTINATION              0x26 //Event could not be proceeded
#define MSG_STATUS_EVENT_NOT_USED                 0x27 //An extern Event has been received which cannot be used here
#define MSG_STATUS_SOURCE_ID_DOESNT_MATCH         0x28 //the sourceID in reactionTab for the extern Event doesn't match
#define MSG_STATUS_INPUT_NUM_DOESNT_MATCH         0x29 //inputNum of Event doesn't match with reactionTab
#define MSG_STATUS_DRIVER_TYPE_NOT_AVAILABLE      0x2a //SendRawData failed
#define MSG_STATUS_DRIVER_NO_ANSWER               0x2b //SendRawData failed
#define MSG_STATUS_WRONG_CMD_PARA                 0x2c //at least one parameter was wrong
#define MSG_STATUS_WRONG_CMD_PARA1                0x2d //para1 exceeds
#define MSG_STATUS_WRONG_CMD_PARA2                0x2e //para2 exceeds
#define MSG_STATUS_WRONG_CMD_PARA3                0x2f //para3 exceeds
#define MSG_STATUS_WRONG_CMD_PARA4                0x30 //para4 exceeds
#define MSG_STATUS_WRONG_CMD_PARA5                0x31 //para5 exceeds
#define MSG_STATUS_UPDATE_NOT_AVAILABLE_ERR       0x32 //could not resolve any of the Download-Commands
#define MSG_STATUS_INCOMPATIBEL_VIRT_LUM          0x33 //"DT8" <--> "normal" Addr not convenient to virtLum
#define MSG_STATUS_EVENT_TAB_ERR                  0x34 //Event could not be found in eventTab
#define MSG_STATUS_RXD_DATA_ENCRYPTED_CRC_ERR     0x35 //CRC of encrypted data failed
#define MSG_STATUS_BLOCK_NUMBER_INVALID           0x36 //invalid block num
#define MSG_STATUS_BLOCKSIZE_NOT_OK               0x37 //blocksize not ok
#define MSG_STATUS_BLOCKSTARTADDR_NOT_ZERO        0x38 //blockStartAddress must be zero according definition
#define MSG_STATUS_RXD_PACKET_ERR                 0x39 //datLength too big, wrong packet
#define MSG_STATUS_LIMIT_REACHED                  0x40 //limit either of virtLum or physLum count exeeds
#define MSG_STATUS_DECRYPTION_FAILED              0x41 //decryption Key 0 or matching failed
#define MSG_STATUS_WRONG_USER_KEY_ID              0x42 //SetUser failed
#define MSG_STATUS_WRONG_NODE_TYPE                0x43 //Not supported by this node type
#define MSG_STATUS_LOCAL_TIME_INVALID             0x44 //missing local time
#define MSG_STATUS_RSP_DATA_OUT_LENGTH_EXCEEDS    0x45 //response data too big for fruity mesh 0.7.1400

//Warnings / Infos
#define MSG_STATUS_IGNORE_SET_REACTION_CONSTRAINTS (0xe0) //informs about reactionTab settings although physDev, scene, timer, sequences
                                                            etc. may not available by the time SetNodeReactionEntry has been received
#define MSG_STATUS_OVERWRITE_INFO                 0xe1 //Overwrite info of any tab settings / phys lum addresses
#define MSG_STATUS_FIRMWARE_RECEIVED_INFO         0xe2 //firmware update completed
#define MSG_STATUS_LUMINAIRE_INCOMPATIBEL_SCENE    0xe3 //Output of a luminaire channel failed: scene channel not configured
```

```
#define MSG_STATUS_MISSING_CONDITION        0xe4 //Command could not be performed, missing condition
#define MSG_STATUS_LR_ACTIVE                0xe5 //Command could be performed, while  light regulation active
```

## 8.10 Appendix J: Physical luminaire type feature

```
#define TYPE_FEATURE_NORMAL     (0) //App-Anzeige “normal”
#define TYPE_FEATURE_DT8_Tc     (1) //App-Anzeige “DT8-Tc”
#define TYPE_FEATURE_DT8_XY     (2) //App-Anzeige “DT8-X/Y”
```

# 8.11 Appendix K: Possible EventType / ActionType Combinations

| | AT_LIGHT_SCENE | AT_DIM_UP_DOWN_X | AT_LIGHT_CONTROL | AT_TIMER | AT_EVENT_TAB_MANIP | AT_SEQUENCE_PROCESS | AT_LIGHT_REGULATION | AT_SLIDER_CONTROL | AT_BEACON_STATE |
|---|---|---|---|---|---|---|---|---|---|
| ET_BUTTON (App) | x | x | | x | | x | x | | x |
| ET_BT_BUTTON_TYPE_x | x | x | | x | x | x | | | x |
| ET_SLIDER (App) | | | | | | | x | x | |
| ET_SLIDER_CONTROL (App) | | | | | | | | x | |
| ET_SENSE_MOVEMENT | x | | x | x | | x | | | |
| ET_SENSE_BRIGHTNESS | | | | | | | | | |
| ET_LIGHT_LEVEL | | | | | | | x | | |
| ET_SYSTEM_TIME | | | | | | | | | |
| ET_TIMER | x | | | | | x | | | |
| ET_REGULATION | | | | | | | x | | |
| ET_EXT_BEACON | | | | | | | | | x |
| ET_SEQUENCE | x | | | | | x | | | |

| | |
|---|---|
| AT_LIGHT_SCENE | Outputs a "Lightlevel"Scene to a virtLum |
| AT_DIM_UP_DOWN_X | outputs one dimm step of brightness of a "Lightlevel"Scene |
| AT_LIGHT_CONTROL | acts on a virtLum regarding activeLL, passiveLL, basicLL: changes brightness of a "Lightlevel"Scene |
| AT_TIMER | enable / disable timerNo in timerTab |
| AT_EVENT_TAB_MANIP | sets / resets activation and/or event forwarding into mesh of a none App event |
| AT_SEQUENCE_PROCESS | (re-)starts a sequence process of "Lightlevel"Scenes |
| AT_LIGHT_REGULATION | defines which virtLum belongs to a light regulation |
| AT_SLIDER_CONTROL | if ET_SLIDER_CONTROL: sets complete channels of a virtLum type independent of its "lightlevel"Scene"; if ET_SLIDER: sets a specific channel according to slider position, needs a "Lightlevel"Scene |
| AT_BEACON_STATE | if ET_EXT_BEACON: sends a SEND_CMD_BEACON_STATE command to Brfirmware; If any BUTTON: changes state |

| | |
|---|---|
| ET_BUTTON (App) | Event delivered from App. Has no entry in eventTab. Has no specific sourceID. Needs user rights |
| ET_BT_BUTTON_TYPE_x | Event received by BRfirmware forwarded to BTfirmware. Has entry in eventTab. Has specific sourceID. Needs user rights. |
| ET_SLIDER (App) | Event delivered from App with a slider position. Has no entry in eventTab. Has no specific sourceID. Needs user rights |
| ET_SLIDER_CONTROL (App) | Event delivered from App with all virtLum type channel values. Has no entry in eventTab. Has no specific sourceID. Needs user rights |
| ET_SENSE_MOVEMENT | Event is generated from own Node and processed. Does nothing if event comes from external Node. |
| ET_SENSE_BRIGHTNESS | Event is generated from own Node and processed. Does nothing if event comes from external Node. |
| ET_LIGHT_LEVEL | Event for internal use. In conjunction with AT_LIGHT_REGULATION in ReactionTab it defines the virtLum which belongs to the light regulation. |
| ET_SYSTEM_TIME | not used for now |
| ET_TIMER | Event is generated from own Node and processed. Needs a TimerTab entry |
| ET_REGULATION | Event can be used for initializing virtLums on other nodes for light regulation. |
| ET_EXT_BEACON | Event received by BRfirmware forwarded to BTfirmware. Has entry in eventTab. |

## 8.12 Appendix L: Command description BR firmware ← BT firmware

PacketType 2

| CommandTypeMsg | Cmd -No | Description | Parameter | Response | Encryption |
|---|---|---|---|---|---|
| GetLocalTime | 0xE0 (224) | Request to BT-firmware to deliver actual Unix UTC local timestamp via SetLocalTime() | none | None<br>The MWAYfirmware answers with Command *SetLocalTime* (Appendix B) | Node key |
| GetMyNodeID | 0xE1 (225) | BT-firmware sets its node DestinationID | none | The MWAYfirmware answers with Command *SetNodeID* (Appendix B) | Node key |
| SetBeaconState | 0xE2 (226) | VSfirmware informs MWAYfirmware about Beacon state as set in AT_BEACON_STATE param 0 | Byte 0: State (0=Off, 1=On, 2 =Toggle) | none | Node key |

# 8.13 Appendix M: Use cases

## 8.13.1 BT Button Type and destinationType

A command from the App or an event from the App or the nodeDevice will deliver destinationType=01 (FROM_TO_MESH) within the MsgHeader.
If delivered from the App no sourceID can be used in the VSfirmware, since this will be dynamically tracked by the MWAYfirmware. An event from the App will not use any eventTab entries. However, if a Bluetooth Button will be used – this will be coupled with the one nodeDevice thru the MWAYfirmware. In this case the MsgHeader must contain destinationType=00 (FROM_LOCAL_NODE) if delivered to VSfirmware.

The following image may show, how a BT_BUTTON_TYPE_1 (coupled with nodeDevice#1) shall react on nodeDevice#2 (which has its own BT_BUTTON_TYPE_1).

reactionTab nodeDevice#1:
not nessacary

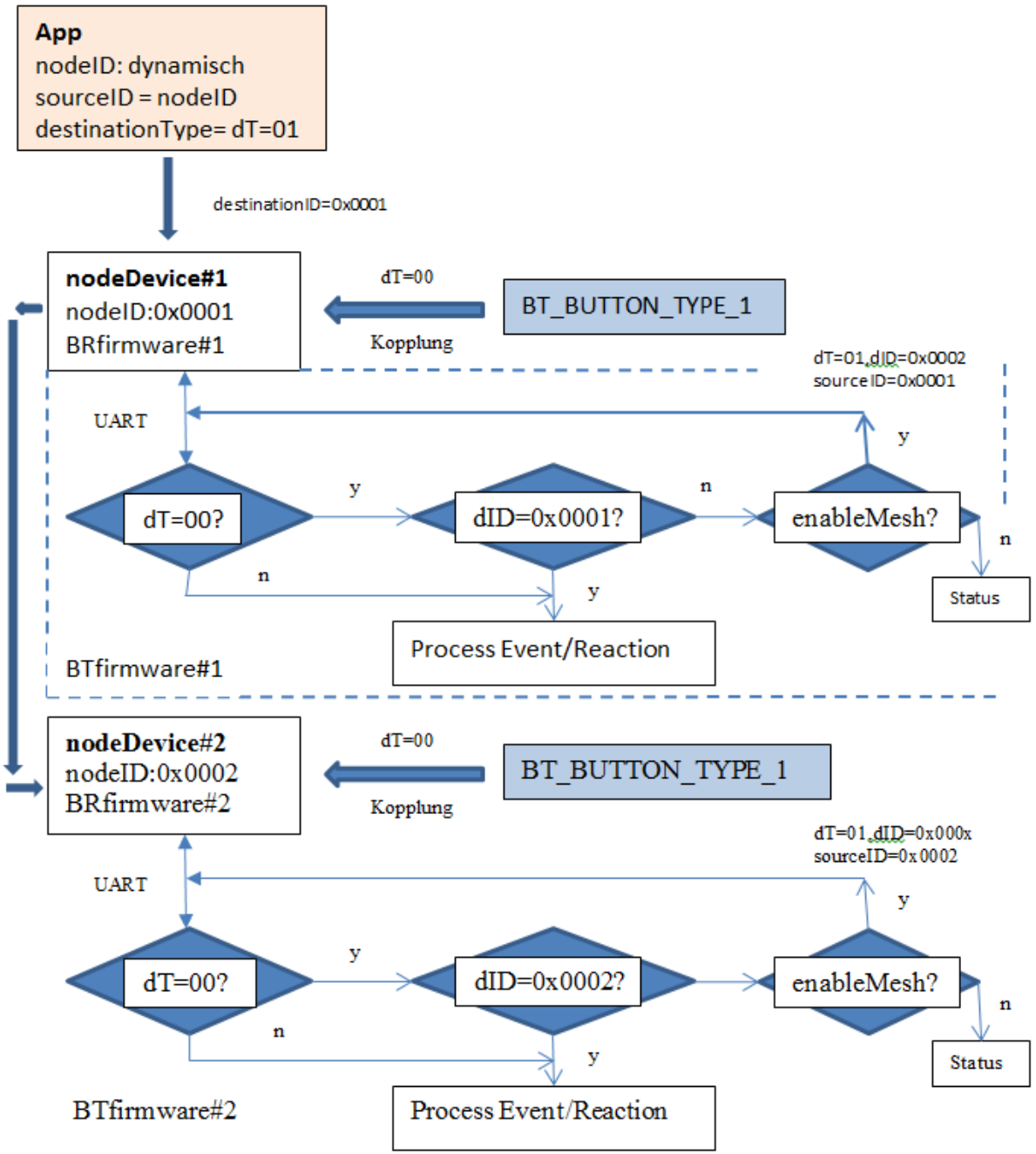
eventTab nodeDevice#1:

```
uint8_t active=0; //each Bit position declares on/off for inputNum0..7
uint8_t enableMesh=0b00000001;//each Bit position declares on/off for inputNum0..7
uint16_t eventTyp=0x0001;
uint16_t para=0x0000;
```

VSfirmware#1 verifies eventTab whether „enableMesh=1“ . If so, a message with header **DestinationType=01** und **DestinationID=0x0002** will be delivered into BT mesh.

MWAYfirmware#2 receives the message and delivers to VSfirmware#2 with sourceID=0x0002, destinationID=0x0002, destinationType=01.
reactionTabnodeDevice#2:

```
uint16_t    eventTyp=ET_BT_BUTTON_TYPE_1
uint16_t    sourceID=0x0001
uint8_t     inputNum=0
uint8_t     actionTyp=1;
uint8_t     userkeyID=0xFF;
uint8_t     param[0..2]={virtLum#2,Scene_x,0}
```


App
nodeID: dynamisch
sourceID = nodeID
destinationType= dT=01
destinationID=0x0001
nodeDevice#1
nodeID:0x0001
BRfirmware#1
dT=00
Kopplung
BT_BUTTON_TYPE_1
dT=01,dID=0x0002
sourceID=0x0001
UART
y
dT=00?
y
dID=0x0001?
n
enableMesh?
n
Status
n
y
Process Event/Reaction
BTfirmware#1
nodeDevice#2
nodeID:0x0002
BRfirmware#2
dT=00
Kopplung
BT_BUTTON_TYPE_1
dT=01,dID=0x000x
sourceID=0x0002
UART
y
dT=00?
y
dID=0x0002?
enableMesh?
n
Status
n
y
BTfirmware#2
Process Event/Reaction

### 8.13.2 Lightcontrol and motion monitoring

A stairway area shall have different luminaire brightness in dependence of a motion detection over time. In order to configure a stairway lighting system we need (for example) to configure one virtLum with its data:

- Create virtLum#1 Typ2 (R,G)
- Set phys lum channel R,G
- Set virtLum#1 Data:

Byte 1: dimSpeed (0..255) (700ms units) = 4
Byte 2: free
Byte 3: free
Byte 4: activeBrightness[step%]=254
Byte 5: passiveBrightness[step%]=127
Byte 6: basicBrightness[step%]=50
Byte 7: min brightness[step%]=0
Byte 8: time active= 5
Byte 9: time passive=7

To turn on virtLum with a Button we need something like this:

- SetScene (sceneNo=2…)
- SetReactionTab(ET_BUTTON,inputNum=1,AT_SCENE,userkeyID=0xff,param1=virtLum#1,param2=LLScene#2)

As soon AT_SCENE will act on virtLum#1 (via Event ET_BUTTON) the MODE_AUTO gets active (because a time active/passive has been configured).

If the lightcontrol passive time has been passed, „basicBrightness“ will set to output. With a signalization of the motion sensor, the virtLum#1 will output “activeBrightness” To obtain this, the following command need to be setup:

- Set ReactionTab ET_SENSE_MOVEMENT, on virtLum 1, Typ 2, AT_LIGHT_CONTROL with Param 4 = moveSenseAct: **MOVE_SENSE_ON_OFF**

Use

- SetInput ET_SENSE_MOVEMENT active

to start a stairway lighting control

## 8.13.3 Lightregulation

The regulation process has to analyze the set setpoint and the feedback to produce the output value of an output device.
Several deviceNodes in a BT-Mesh-Network may own several virtLums. However, for a light regulation we need to define

1) Which virtLums shall act as output device
2) Which deviceNode shall use its brightness sensor for the setpoint value and feedback value
3) How to activate the light regulation

### *8.13.3.1 Define which virtLums shall act as output device*

An „output device“ consist at least in one virtLum. An „output device“ may consist in several virtLums. An „output device“ may consist even in several virtLums on different deviceNodes. However, the VSfirmware works internally with one „LRlum“ (LightRegulation-Luminaire) as an „output device“, which consist at least of one virtLum. To define a simple “output device” we need for example the following commands:

- NodeCreateVirtLum1, Typ2 (R,G)
- Set phys lum1 channel 1(adr1), channel2(adr2)
- Set virtLum1 Data: all NULL
- SetNodeScene2(50%,127r,127g)

For the VSfirmware „LRlum“ consist in the amount of entries in the reactionTab, here virtLum1:

| entry | EventTyp | sourceID | inputNum | actionTyp | userNum | Para1:virtLumNo | Para2:LLscene |
|---|---|---|---|---|---|---|---|
| 5 | ET_LIGHT_LEVEL | 0 | 0 | AT_LIGHT_REGULATION | 255 | 1 | 2 |

- SetNodeReactionEntry(ET_LIGHT_LEVEL,srcID=x,in=0,AT_LIGHT_REGULATION(virtLum1, **LLScene2**))
- SetNodeReactionEntry(ET_LIGHT_LEVEL,srcID=x,in=0,AT_LIGHT_REGULATION(virtLum2, **LLScene3**))
- SetNodeReactionEntry(ET_LIGHT_LEVEL,srcID=x,in=0,AT_LIGHT_REGULATION(virtLumX, **LLSceneX**)) etc.

The relation to a **L**ight**L**evel**S**cene is important for the later use of output value of virtLum1 of LRLum, without changing any color mood.

### *8.13.3.2 Define the output value – define the room brightness the lightregulation shall use*

The output value of the „LRlum“ need to be defined. Meaning each defined virtLum (see above) should be adjusted to users satisfaction of a room brightness. This can be done by simply using command „SetColorControl“ or define a comfortable slider which may not change the color mood, but the brightness:

- SetScene2VirtLum(virtLum=1,sceneNum=2)
- SetNodeReactionEntry(ET_SLIDER,srcID=0, inputNum=2, AT_SLIDER_CONTROL(virtLum1, SLIDER_TYPE_BRIGHTNESS), user=0xFF)

| entry | EventTyp | sourceID | inputNum | actionTyp | userNum | Para1:virtLumNo | Para2:sliderType |
|---|---|---|---|---|---|---|---|
| 3 | ET_SLIDER(21) | 0 | 2 | AT_SLIDER_CONTROL | 255 | 1 | SLIDER_TYPE_BRIGHTNES(0) |

Use e.g. „Send Event ET_SLIDER(inputNum=2,pos[=1..254])” to adjust output value…

### *8.13.3.3 Define the setpoint and start light regulation*

If the output value(s) of the output device LRLum has been setup by the user (see above), the setpoint and the regulation start can be issued by:

- SetInput(ET_SENSE_BRIGHTNESS,active=1,mesh=0,inputNum=0,**para=0xFFFF**) mitgeteilt.

By receiving this command, the VSfirmware will

- Transfer the setpoint to „eventTab[ET_SENSE_BRIGHTNESSindex].para”.
- Start the LightRegulationTask by init LRLum first and output value on LRLum thereafter every 800ms.

## 8.14 Appendix N: CRC8

```
const uint8_t u8CRC8Table[256] = {
 0x00, 0x07, 0x0e, 0x09, 0x1c, 0x1b, 0x12, 0x15, 0x38, 0x3f, 0x36, 0x31, 0x24, 0x23, 0x2a, 0x2d,
 0x70, 0x77, 0x7e, 0x79, 0x6c, 0x6b, 0x62, 0x65, 0x48, 0x4f, 0x46, 0x41, 0x54, 0x53, 0x5a, 0x5d,
 0xe0, 0xe7, 0xee, 0xe9, 0xfc, 0xfb, 0xf2, 0xf5, 0xd8, 0xdf, 0xd6, 0xd1, 0xc4, 0xc3, 0xca, 0xcd,
 0x90, 0x97, 0x9e, 0x99, 0x8c, 0x8b, 0x82, 0x85, 0xa8, 0xaf, 0xa6, 0xa1, 0xb4, 0xb3, 0xba, 0xbd,
 0xc7, 0xc0, 0xc9, 0xce, 0xdb, 0xdc, 0xd5, 0xd2, 0xff, 0xf8, 0xf1, 0xf6, 0xe3, 0xe4, 0xed, 0xea,
 0xb7, 0xb0, 0xb9, 0xbe, 0xab, 0xac, 0xa5, 0xa2, 0x8f, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9d, 0x9a,
 0x27, 0x20, 0x29, 0x2e, 0x3b, 0x3c, 0x35, 0x32, 0x1f, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0d, 0x0a,
 0x57, 0x50, 0x59, 0x5e, 0x4b, 0x4c, 0x45, 0x42, 0x6f, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7d, 0x7a,
 0x89, 0x8e, 0x87, 0x80, 0x95, 0x92, 0x9b, 0x9c, 0xb1, 0xb6, 0xbf, 0xb8, 0xad, 0xaa, 0xa3, 0xa4,
 0xf9, 0xfe, 0xf7, 0xf0, 0xe5, 0xe2, 0xeb, 0xec, 0xc1, 0xc6, 0xcf, 0xc8, 0xdd, 0xda, 0xd3, 0xd4,
 0x69, 0x6e, 0x67, 0x60, 0x75, 0x72, 0x7b, 0x7c, 0x51, 0x56, 0x5f, 0x58, 0x4d, 0x4a, 0x43, 0x44,
 0x19, 0x1e, 0x17, 0x10, 0x05, 0x02, 0x0b, 0x0c, 0x21, 0x26, 0x2f, 0x28, 0x3d, 0x3a, 0x33, 0x34,
 0x4e, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5c, 0x5b, 0x76, 0x71, 0x78, 0x7f, 0x6A, 0x6d, 0x64, 0x63,
 0x3e, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2c, 0x2b, 0x06, 0x01, 0x08, 0x0f, 0x1a, 0x1d, 0x14, 0x13,
 0xae, 0xa9, 0xa0, 0xa7, 0xb2, 0xb5, 0xbc, 0xbb, 0x96, 0x91, 0x98, 0x9f, 0x8a, 0x8D, 0x84, 0x83,
 0xde, 0xd9, 0xd0, 0xd7, 0xc2, 0xc5, 0xcc, 0xcb, 0xe6, 0xe1, 0xe8, 0xef, 0xfa, 0xfd, 0xf4, 0xf3
};

#define proccrc8(u8CRC, u8Data) (u8CRC8Table[u8CRC ^ u8Data])
```

Example:

```
...
 for (uint8_t i=(11+headerData.addressLength); i<(11+headerData.addressLength+datLength); i++)
 {
      crc = proccrc8(crc, UartBufferReadByteWithoutDel(i));
 }
 uint8_t crcData = UartBufferReadByteWithoutDel(11+headerData.addressLength+datLength);
```

```
if (crc != crcData)
{
      // Prüfsumme falsch
}
```

## 8.15 Appendix O: En- & Decryption

```
void XTEAEncipher( uint32_t * v, uint32_t * k)
{
 uint32_t v0=v[0], v1=v[1];
 uint32_t sum=0, delta=0x9E3779B9;
 for(uint8_t i=0; i<64; i++) {
       v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + k[sum & 3]);
       sum += delta;
       v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + k[(sum>>11) & 3]);
 }
 v[0]=v0; v[1]=v1;
}
//------------------------------------------------------------------------------
void XTEADecipher( uint32_t * v, uint32_t * k)
{
 uint32_t v0=v[0], v1=v[1];
 uint32_t delta=0x9E3779B9;
 uint32_t sum = delta * 64;
 for(uint8_t i = 0; i < 64; i++)
 {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + k[(sum>>11) & 3]);
        sum -= delta;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + k[sum & 3]);
    }
 v[0]=v0; v[1]=v1;
}
```

Example:

```
uint32_t nodeKey[4] = {0x12345678,0x12345678,0x12345678,0x12345678};
...
            if (addrData.encryptionType == NODE_KEY_TYP)
            {
                  for (uint16_t i=0; i<datLength; i+=8)
                  {
                        XTEADecipher((uint32_t *)(dataPointer+i), nodeKey);
                  }
            }
```

## 8.16 Appendix P: Sequence table definition

The sequence table in the node can hold up to 64 entries. A sequence consist in presenting several light level scene's, separated by a pause time, on one virtLum (FunctionalGroup). At the end of a sequence a jump to the beginning (loop) and / or an event can be send into mesh. The sequence table can be filled with the command "SetSeqEntry(sceneNo,pause,next)". The FunctionalGroup need to be assigned in the ReactionTab.
The variable "next" controls next action to be performed and consist in next = 0xESxxx xxxx.
E = Event-Bit: setting this bit will send an event into mesh and indicates end of a sequence
S = Stop-Bit: setting this bit will stop the sequence, otherwise will start sequence from beginning (loop)
X = 0..3F, next index in sequence table

Example:Loop

| Sequence name | Seq index | Scene No. | pause | Next | action |
|---|---|---|---|---|---|
| A | 0 | 1 | 3 | 0x01 | Set scene 1 to FunctionalGroup and pause ca. 2 sec |
| A | 1 | 2 | 4 | 0x02 | Set scene 2 to FunctionalGroup and pause ca. 3 sec |
| A | 2 | 5 | 7 | 0x00 | Set scene 5 to FunctionalGroup, pause ca. 10 sec, loop to seqIndex 00 |

Example:Loop and sending an event

| Sequence name | Seq index | Scene No. | pause | Next | action |
|---|---|---|---|---|---|
| A | 0 | 1 | 3 | 0x01 | Set scene 1 to FunctionalGroup and pause ca. 2 sec |
| A | 1 | 2 | 4 | 0x02 | Set scene 2 to FunctionalGroup and pause ca. 3 sec |
| A | 2 | 5 | 7 | **0x80** | Set scene 5 to FunctionalGroup, pause ca. 10 sec, **send an event** ET_SEQUENCE and loop to seqIndex 00 |

Example:stop

| Sequence name | Seq index | Scene No. | pause | Next | action |
|---|---|---|---|---|---|
| A | 0 | 1 | 3 | 0x01 | Set scene 1 to FunctionalGroup and pause ca. 2 sec |
| A | 1 | 2 | 4 | 0x02 | Set scene 2 to FunctionalGroup and pause ca. 3 sec |
| A | 2 | 5 | 7 | **0x40** | Set scene 5 to FunctionalGroup, pause ca. 10 sec, stop sequence |

Example:stop and sending an event

| Sequence name | Seq index | Scene No. | pause | Next | action |
|---|---|---|---|---|---|
| A | 0 | 1 | 3 | 0x01 | Set scene 1 to FunctionalGroup and pause ca. 2 sec |
| A | 1 | 2 | 4 | 0x02 | Set scene 2 to FunctionalGroup and pause ca. 3 sec |

| Sequence name | Seq index | Scene No. | pause | Next | action |
|---|---|---|---|---|---|
| A | 2 | 5 | 7 | **0xC0** | Set scene 5 to FunctionalGroup, pause ca. 10 sec, **send an event** ET_SEQUENCE and stop sequence |

Example:sequence A sends event to start sequence B

| Sequence name | Seq index | Scene No. | pause | Next | action |
|---|---|---|---|---|---|
| A | 0 | 1 | 3 | 0x01 | Set scene 1 to FunctionalGroup and pause ca. 2 sec |
| A | 1 | 2 | 4 | 0x02 | Set scene 2 to FunctionalGroup and pause ca. 3 sec |
| A | 2 | 5 | 7 | **0xC0** | Set scene 5 to FunctionalGroup, pause ca. 10 sec, **send an event** ET_SEQUENCE and stop sequence |

| Sequence name | Seq index | Scene No. | pause | Next | action |
|---|---|---|---|---|---|
| B | 3 | 4 | 3 | 0x04 | Set scene 1 to FunctionalGroup and pause ca. 2 sec |
| B | 4 | 6 | 4 | 0x05 | Set scene 2 to FunctionalGroup and pause ca. 3 sec |
| B | 5 | 9 | 7 | **0x40** | Set scene 5 to FunctionalGroup, pause ca. 10 sec and stop sequence |

Note:

- The Stop-Bit is also set, if “0x7e” is delivered by CMD_SET_SEQUENCE_ENTRY
- A scene has been completed if its fading time has been reached and a pause will be start thereafter.
- Stopping all sequence on a nodeDevice button, is possible via a proper reactionTab entry.
- Stopping a specific sequence on a FunctionalGroup is possible via manual user access (button) on that FunctionalGroup.
- A running sequence onto a FunctionalGroup is in MODE_SEQUENCE
- A FunctionalGroup which is in MODE_SEQUENCE cannot be used for light regulation or light time control (active, passive, basic).
- Sending an ET_SEQUENCE is possible at sequence end position only
- An ET_SEQUENCE can trigger another sequence on a different nodeDevice

Running sequences as a sequence chain on different FunctionalGroup on a single nodeDevice is possible and described in “Timer&DT8.pdf” Docu (P1104-03-S).

## 8.17 Appendix Q: Time calculation according DALI

Several time values are represented by a 1 byte value. The used range for this value is 0 …33.

The following formula has to be used to get the real time value in seconds:

$$t_1 = -1+1.4^x$$

Examples

x = 5
$t_1$ = -1+1.4^5 = 4.37824 seconds

x = 33
$t_1$ = -1+1.4^33 = 66407.7 seconds = 18.4 hours

| x | time[s] |
|---|---|
| 1 | 0,400 |
| 2 | 0,960 |
| 3 | 1,744 |
| 4 | 2,842 |
| 5 | 4,378 |
| 6 | 6,530 |
| 7 | 9,541 |
| 8 | 13,758 |
| 9 | 19,661 |
| 10 | 27,925 |

etc.

## 8.18 Appendix R: Timer Handling

A node Device can hold up to 32 timer settings. A timer expires, if the actual timestamp matches the timestamp configured by the user due to ***SetTimer***. Resolution is 1 minute. A time target setting can be triggered up to a week. A repetitive trigger setting (daily, weekly) is possible.

**Description of SetTimer parameter**

| | |
|---|---|
| Byte 0 | indicates the timer number [1..32] |
| Byte 1-4 | initial local Unix timestamp (UTC). Once a timer elapsed, this value will be updated by VSfirmware if repetition is required |
| Byte 5 | Weekday field where Bit0..Bit6 indicates Monday..Sunday, |
| Byte 6 | Repetitive control: Bit0=daily, Bit1=weekly<br>If either “daily” or “weekly” false, the timer elapsed once within the setting of hour and minute.<br>If “daily” true, the timer elapsed within the setting of hour and minute every day (ignores weekday field).<br>If “weekly” true, each day set in “weekday field” will be used as the time target every week<br>If both “daily” and “weekly” true, weekly will be ignored.<br>Mesh event: send event to mesh: Bit4<br>Setting bit4 forces B2L-App to integrate Timer N in its EventConfiguration. This allows actions on other deviceNodes.<br>Re-setting this bit forces B2L-App to delete the Timer N EventConfiguration and possible reactionTab entry(ies).<br>Enable/Disable Timer: Bit5<br>Setting this bit forces the Timer N to process its task.<br>Re-setting this bit forces the Timer N not to process its task. |

If a repetitive programmed Timer elapsed, the TimerTab entry is always valid for the future.
**If a non-repetitive programmed Timer elapsed, the TimerTab entry needs to be rejected by B2L- and/or T4L-App !**

If the actual timestamp greater or equal the programmed time target, an action will be performed. Several actions maybe processed in this case, if the actual timestamp is invalid for a longer time…

Examples:
**Set a Timer 1 to fixed datetime 02.08.2018 15:00, set mesh event**
“**02.08.2018 15:00**” => 1533222000
Byte 0: 1
Byte 1-4: Time => 1533222000 (dezimal)
Byte 5: 0x0000 0000
Byte 6: xx11 xx00

**Set a Timer 2 to daily at 12:00, no mesh event**
**"01.01.1970 12:00**" => 43200
Byte 0: 2
Byte 1-4: 43200 (dezimal)
Byte 5: 0
Byte 6: xx10 xx01
**Set a weekly Timer 3, every Tuesday and Thursday at 08:00, no mesh event:**
**"01.01.1970 08:00"** => 28800
Byte 0: 3
Byte 1-4: 28800(dezimal)
Byte 5: 0x0000 1010 (Tuesday, Thursday)
Byte 6: xx10 xx10 (x=unused)

## 8.19 Appendix S: Testing the sensor (DUT) in the tester

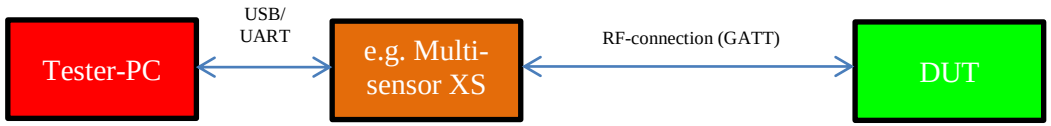
For testing a device with the BlueRangeNode firmware (DUT) in the tester in the VS-production, it is necessary to establish a GATT-connection to make testing of the DUT possible. The data shall be sent from the PC within the tester. There are two possibilities:

1. Establish a connection from a 2nd module (e.g. Multisensor XS) which has already been tested and which working by connecting it via an USB-UART-adapter and establishing a GATT-connection from the PC to the DUT by using the module and sending some user defined commands to check the functionality of the DUT.
2. Establish a GATT-connection from a nRF52-DK (evaluation-kit) which is connected to the tester PC via USB and establish a connection to the DUT with the help of the nRF52-DK.

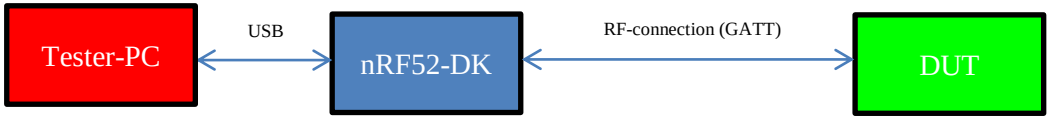
If the solution in point 1 will be used, the config of the BT module inside the modified Multisensor XS in the tester must be adapted by M-Way, so that proper testing can be performed.
If the solution in point 2 will be used, a working configuration for the nRF52-SDK will be supplied by M-Way.

The following graphics show the possible test-cases in the tester:

1. Using a 2nd device (e.g. modified Multisensor XS):




2. Using a "nRF52-DK" (evaluation-kit) in the tester:




The following ASCII-commands have been defined for establishing a connection or disconnecting a modified Multisensors XS or nRF52-DK via GATT from the DUT:

| Msg | Description | Parameter | Response | Encryption |
|---|---|---|---|---|
| `MakeGattConnection` | Establish a GATT-connection to the DUT | BLE-GAP-Address, Serial number, Node Key. | OK (Connection successful)<br>FAIL (Connection not successful) | none |
| `DisconnectGatt` | Disconnect an existing connection from the DUT | none | OK (Disconnected successfully) | none |

If a GATT-connection is established, the defined events and reactions from the document shall be valid (like an APP-connection to the DUT).
VS prefers the solution with the nRF52-DK, because it is already used for programming the BT module of the DUT.

# 8.20 Appendix T: StartUp nodeDevice and lightlevel on a virtLum

### 8.20.1 Power Ready

The MWAYfirmware must wait for its bluetooth communication until its detection of ET_POWER_READY (sending by VSfirmware). This becomes necessary because of power absorption if bluetooth communication becomes true.
The VSfirmware checks its µC-voltage-power first and will send ET_POWER_READY as soon power has been stabilized.
While ET_POWER_READY has not been received by the MWAYfirmware, the MWAYfirmware itself should request this event via ET_POWER_READY every 100ms. On request, the VSfirmware may send either ET_POWER_ READY(0) or ET_POWER_READY(1).
If the event ET_POWER_READY(1) could not be received within 2 seconds by the MWAYfirmware, the controller may start its bluetooth communication nevertheless.

### 8.20.2 Default Scene

Create a default LLScene#1 if not exist. LLScene#1 can of course be overwritten by the user.
#define DEFAULT_BRIGHTNESS_SCENE (127)
#define DEFAULT_TIM2TARGET_SCENE (0)
#define DEFAULT_CHAN0_SCENE (254)
#define DEFAULT_CHAN1_SCENE (254)
#define DEFAULT_CHAN2_SCENE (254)
#define DEFAULT_CHAN3_SCENE (254)
#define DEFAULT_CHAN4_SCENE (254)
#define DEFAULT_CHAN5_SCENE (254)

### 8.20.3 Virtual Luminaire, Functional Group (FG)

Each FG which has been channel configured, may react in its lightlevel, depending of the action within the reactionTab. The output lightlevel of a FG is prioritized as shown in the following tab:

| Prio | Event | Scene num, shown in | % - Lightlevel as shown in | Mode setting | Init activeTime |
|---|---|---|---|---|---|
| 1 | Timer (if < localTime), light regulation start | reactionTab entry | eventTab[LIGHT_REGULATION] | auto | y |
| 2 | Timer (if < localTime), auto switch | reactionTab entry | FG parameter activeLL | auto | y |
| 3 | Timer (if < localTime), manual switch | reactionTab entry | sceneTab: sceneLL | manual | n |
| 4 | Timer (if < localTime), Sequence, start | reactionTab entry | sequenceTab: sceneLL | sequence | n |
| 5 | Timer / Auto switch | reactionTab entry | FG parameter basicLL | auto | n |
| 6 | Movement sense | reactionTab entry | FG parameter basicLL | auto | n |
| 7 | Brightness sense, light regulation | reactionTab entry | eventTab[LIGHT_REGULATION] | auto | y |

| | | | | | |
|---|---|---|---|---|---|
| 8 | Manual switch | reactionTab entry | sceneTab: sceneLL | manual | n |
| 9 | Sequence switch | reactionTab entry | sequenceTab: sceneLL | sequence | n |
| 10 | FG exist, but no reaction programmed | - | 0 % | manual | n |

### 8.20.4 Changing reactionTab during user event configuration

After power up severel FG's may have its specific output lightlevel. A change of the reactionTab afterwards may automatically change the outputlevel of the specific FG in the same way as stated in the tab above.

## 8.21 Appendix U: virtual luminaire data

Following data need to be placed in the APP during creation of a virtual luminaire type:
**Byte 0: Number Virtual Luminaire(1..16)**
User selection of the virtual luminaire number 1..16 of the own node. We start with 1, since internally the VSfirmware uses "0" for internal checks. However, this applies for most other vars too.

**Byte 1: dimSpeed (0..255)**
Needed for manual dimming up/down. Internally this number will be used in units of 700ms. One step down uses a deltaDALI value to output next lightlevel. "deltaDali" corresponds to "stepCount" (see Appendix C) within 700ms. If the user wants to delay further more this behaviour, he can do it with "dimSpeed". A "1" will show up the lightlevel within 1400ms. A "10" will show up the next lightlevel smoothly within 70700ms.

**Byte 2: reserved1**
**Byte 3: reserved2**

**Byte 4: active brightness[step%]**
A virtual luminaire belongs to a light regulation if "**light regulation active = 1**" and if it is in "active" state. The user may select a number between 0..100 (%) while the MWAYfirmware sends 0..254 (step%) to the VSfirmware.

**Byte 5: passive brightness[step%]**
While a virtual luminaire is in "active" state for "time active" it will change its "active brightness" to "passive brightness". Also it will be rejected for a light regulation if so. A movement sensor sense may recall the "active" state.

**Byte 6: basic brightness[step%]**
While a virtual luminaire is in "passive" state for "time passive" it will change its "passive brightness" to "basic brightness". Also it will be rejected for a light regulation if not already done so. A movement sensor sense may recall the "active" state.

**Byte 7: min brightness[step%]**
There is a “min brightness” which will be evaluated from all the DALI-driver which are configured within the virtLum type and its physical luminaire connections. The VSfirmware may retrieve this value by “GetVirtLumData” as soon the virtLum has been fully configured with its phys channels. The user may increase the value in the App but not decrease.

**Byte 8: time active**
A value “0” will never reach a “passive” state. A value > 0 will switch to “passive” state after the “active” time has been expired (otherwise another event will retrigger the “active” time)

**Byte 9: time passive**
A value “0” will never reach a “basic” state. A value > 0 will switch to “basic” state after the “passive” time has been expired (otherwise another event will retrigger the “active” time)

## 8.22 Appendix V: moveSenseAct

This variable is used as a parameter for AT_LIGHT_CONTROL. If a movement sensor shall act on light time / light regulation with a virtLum.
An ET_ SENSE_MOVEMENT Event may act in conjunction with AT_LIGHT_CONTROL in various ways:

**#define MOVE_SENSE_UNKOWN 0**
Description: action not initialized / movement sensor may not available

**#define MOVE_SENSE_ON_OFF 1** (AKTOR_FUNKTION_SENSOR_BEWEGUNG_ON_OFF)
Description: sets lightlevel to activeLL, retriggers activeTime

**#define MOVE_SENSE_OFF 2** (AKTOR_FUNKTION_SENSOR_BEWEGUNG_OFF)
Description: same as MOVE_SENSE_ON_OFF, but: if basicLL has been reached no action should be performed on the virtLum.

**#define MOVE_SENSE_PASSIV 3** (AKTOR_FUNKTION_SENSOR_BEWEGUNG_PASSIV)
Description: switches virtLum to "passivLL", if virtLum outputs basicLL. Retriggers passiveTime.

**#define MOVE_SENSE_MANUAL 4** (AKTOR_FUNKTION_LICHTWERT)
Description: switches virtLum to “activeLL“, turns off light time control.

## 8.23 Appendix W: Search modes

**#define CLEAR_AND_SEARCH 0**

Clear & Search:

- resets all virtLums,
- erases all internal LED driver group addresses,
- resets all DALI driver short addresses,
- starts generic search of connected LED driver and saves its short address

Command "`GetPhysLumSearch`" informs about all available phys lum driver in a 64-Bit variable.

**#define ADD_ONLY_UNADDRESSED 1**

Search only New:

- starts generic search of connected LED driver which have not been assigned to a short address
- saves an eventually new short address
- does not change a virtLum configuration

Command "`GetPhysLumSearch`" informs about new available phys lum driver channels in a 64-Bit variable.

**#define ADD_ONLY_UNADDRESSED_DOUBLE 2**

Keep & Search:

- starts generic search of connected LED driver which have not been assigned to a short address
- saves an eventually new short address
- checks if a driver contains a double assigned physical address (this is possible, if a user connects a new physical lum which owns already the same physical address of another one). If so, this channel address will be reset and new addresses will be given. This follows that a virtLum could be assigned to different physical channel address prior of this search!

Command "`GetPhysLumSearch`" informs about new available phys lum driver channels after last search in a 64-Bit variable. If double address found, this address will be reflected as "new" since it cannot be assigned to an eventually created previous virtLum with that address.

Command „GetNodePhysLum" informs about each available physical driver channel addresses.

## 8.24 Appendix X: Slider Type

A slider type indicates the handling / calculation of a virtLum's phys channel.

A reactionTab entry ET_SLIDER, inputNum should be connected to an action AT_SLIDER_CONTROL(virtLum, sliderTyp). The entrance of an ET_SLIDER(pos, inputNum) would than react on the defined sliderTyp as follows:

**#define SLIDER_TYPE_BRIGHTNESS 0**
The brightness of the specified virtLum (meaning all its available channels) will be adjusted according to the brightness slider position.

**#define SLIDER_TYPE_SINGLE 1**
The R-channel of the specified virtLum will be adjusted according to its position.

**#define SLIDER_TYPE_TUNABLE_WHITE 2**
The RG-channel of the specified virtLum will be calculated in VSfirmware according to its position:
Position = 127 →R=127,G=127
Position = 128 →R=126,G=129
Position = 100 →R=100,G=154
Position = 0 → R=0, G=254
Position = 254 → R=254, G=0

**#define SLIDER_TYPE_RGB 3**
The RGB-channel of the specified virtLum will be calculated according to its position:
Not available yet

**#define SLIDER_TYPE_RGBW 4**
The RGBW-channel of the specified virtLum will be calculated according to its position:
Not available yet

**#define SLIDER_TYPE_RGBWA 5**
The RGBWA-channel of the specified virtLum will be calculated according to its position:
Not available yet

**#define SLIDER_TYPE_RGBWAF 6**
The RGBWAF-channel of the specified virtLum will be calculated according to its position:
Not available yet

**#define SLIDER_TYPE_LIGHT_REGULATION 7**
As SLIDER_TYPE_BRIGHTNESS: For each virtLum detected in reactionTab with ET_LIGHT_LEVEL and AT_LIGHT_REGULATION

A reactionTab entry ET_SLIDER_CONTROL connected with an action AT_SLIDER_CONTROL ignores the sliderTyp but sets each available virtLum channel according to its ET_SLIDER_CONTROL parameter. This is an adaption of the command “SetColorControl”.

| | **Event ET_SLIDER** | **Event ET_SLIDER_CONTROL** |
|---|---|---|
| **SLIDER_TYPE_BRIGHTNESS** | Ok (act on brightness level of FunctionalGroup(s) which owns the scene number) | Untested (act on all channels, no scene relation) |
| **SLIDER_TYPE_SINGLE** | Ok (act on R-channel (channel 1) of FunctionalGroup(s) which owns the scene brightness level) | Untested (act on R-channel in conjunction with brightness level, no scene relation) |
| **SLIDER_TYPE_TUNABLE_WHITE** | Ok (act on RG-channel (channel 1&2) of FunctionalGroup(s), no scene relation) | Untested (act on RG-channel in conjunction with brightness level, no scene relation) |
| **SLIDER_TYPE_RGB** | Not implemented | Ok (act on RGB channels, no scene relation) |
| **SLIDER_TYPE_RGBW** | Not implemented | Ok (act on RGBW channels, no scene relation) |
| **SLIDER_TYPE_RGBWA** | Not implemented | Not implemented, from App |
| **SLIDER_TYPE_RGBWAF** | Not implemented | Not implemented, from App |

## 8.25 Appendix Y: Bootloader

A new VSfirmware consist in a binary encrypted file, which will be generated by VS. This “BTSensorV00000000.bin” file will be delivered to M-Way to generate a new Tablet-/Handy “LINA”-App. This binary file consist in

| Address | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000000 | 00 | 00 | 00 | 00 | 00 | 00 | 01 | 10 | 00 | 01 | e0 | 00 | 01 | e0 | 00 | 00 |
| 00000010 | 01 | 00 | 00 | 00 | 00 | 00 | ae | e5 | 06 | 23 | ed | 5c | 6e | 7b | 97 | 65 |
| 00000020 | 78 | 63 | 67 | 15 | 55 | 97 | 12 | 4d | 92 | fc | 50 | ed | d0 | 4a | 12 | 4d |
| 00000030 | 92 | fc | 50 | ed | d0 | 4a | 12 | 4d | 92 | fc | 50 | ed | d0 | 4a | 1e | 1a |
| 00000040 | 3e | fa | 51 | aa | 37 | 00 | 12 | 4d | 92 | fc | 50 | ed | d0 | 4a | ca | 57 |
| 00000050 | 8f | 92 | d0 | d2 | 93 | 4b | 74 | b4 | 6b | b5 | 12 | c3 | ce | e4 | 74 | b4 |
| 00000060 | 6b | b5 | 12 | c3 | ce | e4 | 5c | f2 | 7c | 67 | 63 | 59 | c9 | 57 | 7a | c4 |
| 00000070 | e4 | 51 | 1f | 96 | ac | 18 | d0 | 0c | b1 | 27 | ad | e8 | 1d | 76 | 57 | 71 |
| 00000080 | 07 | c2 | ac | 90 | 97 | 31 | 01 | 36 | b1 | 84 | fd | fd | 2e | 97 | eb | 65 |
| 00000090 | cd | d1 | 1e | ec | 92 | 78 | 99 | 3e | d3 | 4a | 0d | 74 | 34 | ec | 2e | 75 |
| 000000a0 | cf | fb | c0 | ae | 77 | 58 | 61 | 85 | bc | 4c | 11 | 9b | d1 | 7b | 1e | 1a |
| 000000b0 | 3e | fa | 51 | aa | 37 | 00 | 74 | b4 | 6b | b5 | 12 | c3 | ce | e4 | 74 | b4 |
| 000000c0 | 6b | b5 | 12 | c3 | ce | e4 | 46 | c9 | 81 | eb | 04 | ee | 90 | 0c | 5c | bb |
| 000000d0 | 9a | f0 | c4 | b4 | bc | 70 | 47 | e4 | b6 | 29 | 3c | 20 | bc | 5c | 8f | 2a |
| 000000e0 | 5b | 50 | ad | ab | eb | 07 | b0 | 3a | 93 | ce | 37 | e7 | 65 | 51 | ad | 7d |
| 000000f0 | 6d | 08 | 48 | 21 | 41 | 28 | cf | c4 | eb | cb | d2 | e1 | ae | 13 | 18 | a4 |
| 00000100 | d8 | b7 | 78 | ba | b0 | 20 | ff | 78 | 02 | a4 | 1a | 8f | 9a | 77 | da | 57 |
| 00000110 | 14 | 1b | 64 | 90 | 23 | 3a | 26 | 6c | 00 | 01 | e0 | 00 | 01 | e0 | 00 | 01 |
| 00000120 | 01 | 00 | 00 | 00 | 00 | 00 | a1 | aa | b4 | 03 | bd | 0d | dc | 56 | 1c | b6 |

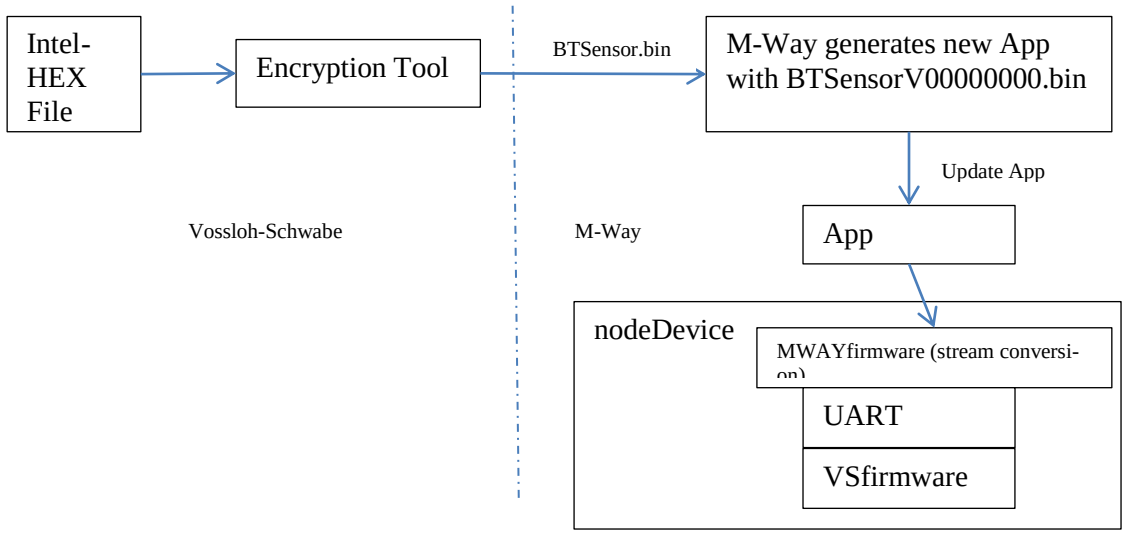
Byte 0..3:
4 Bytes version number
Byte 4..7:
4 Bytes block length = page_size 256 + firmware_size 4 + blocks_total 2 + current_block 2 + block_size 2 + block_startaddr 4 + CRC 2 = 272 == 0x00000110
Byte 8..nn
All data as a stream
At the end "EEEE" 0x45454545 as „EOF“

The encrypted VSfirmware File need to be delivered via command “`TransmitFirmwareData`” and looks like this:
00000000 00000110 (unverschlüsselt)
0001e000 01e0 0000 0100 00000000 (unverschlüsselt) aee50623…”CCCC”
0001e000 01e0 0001 0100 00000000 (unverschlüsselt) a1aab403…”CCCC”
…
45454545

The data stream needs to be delivered by “TransmitFirmwareData”, for example:

TransmitFirmwareData (Firmware Size = 0001e000, Blocks Total = 01e0,Current Block = 0000, Block Size=0100, Block Start Address = 00000000, encrypted Block Data = aee50623…CCCC)
TransmitFirmwareData (Firmware Size = 0001e000, Blocks Total = 01e0,Current Block = 0001, Block Size=0100, Block Start Address = 00000000, encrypted Block Data = a1aab403…CCCC)
…

Note that Block Start Address is reserved and contains “00000000”.
The version number “00000000” need not to be delivered since it indicates a change, the file is organized.



# 8.26 Appendix Z: VS Internal Commands for Tester

We defined 2 commands to handle the function test of our devices inside the function tester.
They are only used for tester or debug purposes and needs no configuration !!!

## A. SendRawData (type, data1, data1+x … data7)

This command is used to send data over the lighting control interface of the device (ex. DALI) .

Byte 0 describes the type of data (example 0x00 = DALI)
Currently, we have only the type 'DALI'. More types can be defined in future.

For Type 'DALI', we use a similar protocol as our VS DALI Protocol, which is further described in document 'VS DALI protocol'.

**Normal DALI command 'Off', broadcast**

| Bit count<br>Decimal / hexadecimal | Repeat (after x ms) | Answer expected after x ms | DALI Byte 1 | DALI Byte 2 | Reserved |
|---|---|---|---|---|---|
| 16 / 0x10 | 0 | 0 | 0xFE | 0x00 | |

**Repeated DALI command 'Reset', broadcast**

| Bit count | Repeat (after x ms) | Answer expected after x ms | DALI Byte 1 | DALI Byte 2 | Reserved |
|---|---|---|---|---|---|
| 16 / 0x10 | 0x65 | 0 | 0xFF | 0x20 | |

**DALI command with Answer (Query) : 'QueryStatus', broadcast**

| Bit count | Repeat (after x ms) | Answer expected after x ms | DALI Byte 1 | DALI Byte 2 | Reserved |
|---|---|---|---|---|---|
| 16 / 0x10 | 0 | 0x65 | 0xFF | 0x90 | |

**DALI Answer (Query) , received as BT_Response**

| MsgStatus | Bit count | Time till answer (ms) | Error code | DALI Answer Byte |
|---|---|---|---|---|
| | 8 / 0x08 | 0x75 | 0x00 | 0xFF |

**DALI Error codes:**

0 Message (no error)
1 Time Out
2 Frame Error
3 Overlayed Answer

## B. GetSensorData () -> Response : MsgStatus, MovementStatus (1Byte), Brightness (2 Bytes)

The command GetSensorData requests the actual state of the movement sensor and brightness value of the brightness sensor and send it back by response.
This will need no special preparation / configuration of the sensor before !

Byte 0: CmdNo
Byte 1:MsgStatus
Byte 2: updateLevel MSB
Byte 3: updateLevel LSB
If MULTISENSOR:
Byte 4: Movement State
Byte 5..6: Brightness value
If CONNECT
Byte 4: 0x0000 0nn0
Bit 1: Button 1 pressed
Bit 2: Button 2 pressed
Byte 5..6: 0

## C. UART-Test-Frame

This UART-frame can be used to test UART communication from/to BT module.