CAN and General

BLF Logging Format

[Document Type]

Version 1.48 of 2020-03-30

Status	Completed
Publisher	Vector Informatik GmbH
	 Note: Hardcopy documents are not subject to change management.

Document Management

Revision list

Version	Date	Editor	Section	Changes, comments
0.1	2008-06-09	Gey	All	Initial version created.
0.2	2008-06-17	Gey	All	Rework after review
1.0	2008-06-19	Gey	All	Added BL_OBJ_* values to the object types. Added VBLObjectHeader2
1.2	2008-09-24	Ae	3.1	Extended CAN message flags
1.3	2009-05-18	Gia	1	Added Disclaimer
1.4	2010-01-15	Sc	3.1	New member mObjectVersion in VBLObjectHeader and VBLObjectHeader2
1.5	2010-08-12	Jr	3.13	Added example for VBLEthernetFrame
1.6	2010-10-26	Sha	3.2; 3.3	Added VBLCANMessage2
1.7	2010-11-29	Hb	3.5	VBLCANErrorFrameExt extensions
1.8	2010-12-21	Hb	3.5	VBLCANErrorFrameExt extensions
1.9	2011-01-19	Sha	3.2; 3.3	Hints added to VBLCANMessage and VBLCANMessage2 description
1.10	2011-04-07	Mp	3.17	Added Comment event (for comments in Trace Window)
1.11	2011-05-20	Jr	3.15, 3.16	Added WLAN events
1.12	2011-08-29	Jr	3.15	Modified signal strength of WLAN event
1.13	2011-09-08	Sha	3.3	Clarification to CAN message length
1.14	2011-10-24	Mp	3.20	Added global marker event (for global markers)
1.15	2012-03-01	Wwi	3.21, 3.22	Added AFDX events
1.16	2012-05-07	Wwi	3.21	AFDX flag bit enumeration updated
1.17	2012-08-23	Chk	3.1.2,3.4	Added CAN FD message flag description, Added VBLCANFDMessage description
1.18	2012-09-06	Chk	3.4	VBLCANFDMessage format modified
1.19	2012-09-06	Chk	3.4	VBLCANFDMessage format modified
1.20	2012-10-22	Gia	3.19	Some modifications for VBLAppText
1.21	2013-04-15	Fsi	3.4, 3.7	CAN FD BLF Logging
1.22	2013-04-23	Jr	3.17. 3.18	Added Ethernet status and Rx error
1.23	2013-04-23	Hb	3.4	CAN FD BLF Logging
1.24	2013-07-12	Hb	3.4	CAN message-flags added
1.25	2013-10-15	Chk	3.4,3.7	Data length for CAN remote frames
1.26	2014-02-13	Hb	3.4, 3.7	Clarifications
1.27	2014-03-04	Wwi	3.25–3.29	Added AFDX status event and new bus statistic
1.28	2014-12-10	Jmi	3.18	Added Ethernet bus statistic event
1.29	2014-12-10	Hb	3.4, 3.7	Extended CAN FD events

Version	Date	Editor	Section	Changes, comments
1.30	2014-12-11	Wwi	3.31-3.35	Added AFDX error event and all A429 events
1.31	2015-01-21	Uru	3.32, 3.35	A429 message and A429 Statistic format modified
1.32	2015-01-26	Rue	3.14	mRepresentation flag for system variable
1.33	2015-01-28	Lt	3.1.4	mClientIndex for internal use
1.34	2015-02-13	Chk	3.6	Add Ack Error failure code
1.35	2015-02-17	Chk	3.4	Changed description of mFlag Bit 19, because differ to source code description
1.36	2015-04-01	Lke	3.36	Added test structure events
1.37	2015-04-07	Chk	3.4, 3.7	Replace EDL by FDF
1.38	2015-04-23	Lke	3.36	Test structure events modified
1.39	2015-09-01	Lke	3.36	Corrected description for VBLTestStructure::mUniqueNo
1.40	2016-02-15	Wwi	3.25–3.29	Improve description for AFDX/A429
1.41	2016-06-22	Jr	3.17	Moved Ethernet events to own document
1.42	2016-10-12	Srj	3.34-3.36	Added queue events
1.43	2017-02-24	Mom	All	CI and layout
1.44	2017-03-07	Vrd	3.1.6, 3.18	Added Trigger Condition event
1.45	2018-07-20	Chk	3.13	Add Reset/Bit Timing Changed event
1.46	2018-10-26	Chk	3.4	Correct description of CAN-FD event flags
1.47	2020-02-07	vsn	3	API has changed to standard types, e.g. uint32_t instead of DWORD, added libbinlog.so
1.48	2020-03-30	Bma	3.12	Added AppText type BL_APPTEXT_ATTACHMENT

VECTOR

Contents

1 Disclaimer

Severability clause - Restrictions for the usage of Vector logging data formats outside of Vector products

The format specification / access functions for the Vector BLF and ASC logging data formats are made available under the restrictions and conditions cited hereafter.

Please note that Vector Informatik neither gives any guarantee nor assumes any liability beyond compulsory legal regulations for the BLF or ASC logging format respectively as well as for the access functions to the single objects.

Vector Informatik disclaims all liability for errors which might be contained in the access functions or the format specification itself.

Vector Informatik does neither provide support for the integration into your software nor for problems occurring inside your software on the customer side.

Beyond that Vector Informatik reserves the right to change the BLF or ASC data format respectively anytime without prior notification. Therefore, the compatibility of the format is not ensured.

2 Overview

The document specifies the format of CAN events and general objects in the CANoe/CANalyzer BLF logging. The described structures can be used to read and write BLF logging files using the binlog.dll or libbinlog.so, which can be found in the CANoe/CANalyzer User Data folder:

<UserDataFolder>\Programming\BLF_Logging

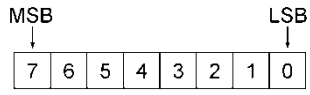
3 Format Description

3.1 Common Data Types

3.1.1 CAN Message Flags

The following flags are valid for the **mFlags** members of CAN objects.

1. Direction of CAN frame (DIR)
2. Remote Transmission Request (RTR).
3. Single wire operation (NERR)
4. Wake Up Message (high voltage) (WU)



RTR (Bit 7)		WU (Bit 6)		NERR (Bit 5)		DIR (Bit 3-0)	
0:	No RTR	0:	No WU	0:	No NERR	0:	RX
1:	RTR	1:	WU	1:	NERR	1:	TX

Use the following macros to determine the values of RTR, WU, NERR and DIR:

```
#define CAN_MSG_DIR( f)          ( uint8_t)(   f & 0x0F)
#define CAN_MSG_RTR( f)          ( uint8_t)( ( f & 0x80) >> 7)
#define CAN_MSG_WU( f)           ( uint8_t)( ( f & 0x80) >> 6)
#define CAN_MSG_NERR( f)         ( uint8_t)( ( f & 0x80) >> 5)
```

Use the following macro to set the values of RTR and DIR:

```
#define CAN_MSG_FLAGS( dir, rtr) ( uint8_t)( ( ( uint8_t)( rtr & 0x01) << 7) | \
                                               ( uint8_t)( dir & 0x0F))
```

Use the following macro to set the values of RTR, WU, NERR and DIR:

```
#define CAN_MSG_FLAGS_EXT( dir, rtr, wu, nerr) \
                                  ( uint8_t)( ( ( uint8_t)( rtr  & 0x01) << 7) | \
                                              ( ( uint8_t)( wu   & 0x01) << 6) | \
                                              ( ( uint8_t)( nerr & 0x01) << 5) | \
                                                ( uint8_t)( dir  & 0x0F))
```

To set the RTR flag to **true** and the frame direction to a send frame, execute:

```
uint8_t flags = CAN_MSG_FLAGS( 1, 1)
```

To set the RTR flag to **true**, the WU flag to **true**, the NERR flag to **true** and the frame direction to a send frame, execute:

```
uint8_t flags = CAN_MSG_FLAGS_EXT( 1, 1, 1, 1)
```

3.1.2 Driver Error Codes

The following values are error codes valid for CAN driver error information

Value	Description
0	timeout during board initialization
1	no events in the rx queue / no event available for dvGetEvent

Value	Description
2	tx queue full, tx request refused
3	unknown Controller-Nr.
4	timeout during command
5	DPRAM-Overflow
6	not allowed event in dvPutCommand
8	driver detected another hardware (see CANIB)
9	parameter error in dvMeasureInit
10	parameter error in dvMeasureInit and dvPutCommand
11	not (yet) implemented function in this version of driver
12	82526: no access to imp
14	last msg wasn't transferred
100	unknown send id (FullCAN only)
101	rx queue overrun
102	chip state busoff
103	chip state error passive
104	chip state error active
105	rx register overrrun (BasicCan only)
106	at bootup the firmware couldn't access the controller
107	no valid dpram address in dvBoardInit
108	no interrupt from CANIB received
109	wrong modulnumber
110	wrong pointer to source buffer
111	address > CANIB_SRAM_SIZE
112	address + size > CANIB_SRAM_SIZE
113	CAN-Nr. <> 0 && CAN-Nr. <> 1
114	FIFO-Entry > 16 Byte

Value	Description
115	see drvspec
codes 200..455 are reserved for CANIB-Driver	
217	Enable CPU not successful
218	Set of TR-Status Bit not successful
219	Halt command not successful
220	Halt command not successful
221	Reset command not successful
222	Reset command not successful
232	Timeout waiting for Data from FIFO
248	Unknown command
249	Unknown function
250	Wrong parameter format
251	Wrong parameter
252	OK-message while waiting for data
455	Unknown answer from CANIB

3.1.3 VBLObjectHeaderBase

Description: Object header base structure.

Parameter	Type	Description
mSignature	uint32_t	Object signature, must be `BL_OBJ_SIGNATURE`.
mHeaderSize	uint16_t	Size of header in bytes, set this member to `sizeof(VBLObjectHeader)` or `sizeof(VBLObjectHeader2)` depending on the object header type used for the object.

Parameter	Type	Description
mHeaderVersion	uint16_t	Version number of object header. Set this member to 1 if the object has a member of type `VBLObjectHeader`. Set this member to 2 if the object has a member of type `VBLObjectHeader2`.
mObjectSize	uint32_t	Object size in bytes.
mObjectType	uint32_t	Object type (`BL_OBJ_TYPE_*`).

3.1.4 VBLObjectHeader

Description: Object header. Version 1.

Parameter	Type	Description
mBase	VBLObjectHeaderBase	Common object header base. See 3.1.3.
mObjectFlags	uint32_t	Unit of object timestamp. Following values are possible: 1: Object time stamp is saved as multiple of ten microseconds (`BL_OBJ_FLAG_TIME_TEN_MICS`) 2: Object time stamp is saved in nanoseconds. (`BL_OBJ_FLAG_TIME_ONE_NANS`)
mClientIndex	uint16_t	For internal use.
mObjectVersion	uint16_t	Object specific version, has to be set to 0 unless stated otherwise in the description of a specific event.
mObjectTimeStamp	uint32_t	Time stamp of this object in the unit specified in mObjectFlags.

3.1.5 VBLObjectHeader2

Description: Object header. Version 2.

Parameter	Type	Description
mBase	VBLObjectHeaderBase	Common object header base. See 3.1.3.

Parameter	Type	Description
mObjectFlags	uint32_t	Unit of object timestamp. Following values are possible: 1: Object time stamp is saved as multiple of ten microseconds (BL_OBJ_FLAG_TIME_TEN_MICS) 2: Object time stamp is saved in nanoseconds. (BL_OBJ_FLAG_TIME_ONE_NANS)
mTimeStampStatus	uint8_t	Bit field. The bits have the following meanings: Bit 0: Determines whether original timestamp member is valid (1) or not (0). Bit 1: Timestamp is generated by software (1) or by hardware (0). Bit 5: This bit has protocol specific meaning.
mReserved1	uint8_t	Reserved, must be 0.
mObjectVersion	uint16_t	Object specific version, has to be set to 0 unless stated otherwise in the description of a specific event.
mObjectTimeStamp	uint32_t	Time stamp of this object in the unit specified in mObjectFlags.
mOriginalTimeStamp	uint32_t	Original timestamp in the unit specified in mObjectFlags.

3.1.6 VBLVarObjectHeader

Description: Extendible variable length object header.

Parameter	Type	Description
mBase	VBLObjectHeaderBase	Common object header base. See 3.1.3.
mObjectFlags	uint32_t	See 3.1.4
mObjectStaticSize	uint16_t	Size of the mStatic part of the object
mObjectVersion	uint16_t	See 3.1.4
mObjectTimeStamp	uint32_t	See 3.1.4

3.2 Obsolete Types

3.2.1 VBLCANMessage

Description: CAN data or CAN remote frame received or transmitted on a CAN channel.

Corresponding object type: BL_OBJ_TYPE_CAN_MESSAGE

Obsolete object. Used up to CANoe/CANalyzer version 7.2

Hint: This object is also used in later CANoe/CANalyzer versions when the following flag is set in the CAN.INI file.

[CAN]

BLFFormat_Compatible_with_7_2_and_ealier = 1

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Channel the frame was sent or received.
mFlags	uint8_t	See 3.1.1
mDLC	uint8_t	Data length code of frame (number of valid data bytes, max. 8)
mID	uint32_t	Frame identifier.
mData[8]	uint8_t	CAN data bytes

3.3 VBLCANMessage2

Description: CAN data or CAN remote frame received or transmitted on a CAN channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_MESSAGE2`

Object available starting from CANoe/CANalyzer version 7.5

Hint: This object is used only when the following flag is NOT set in the CAN.INI file, otherwise the object VBLCANMessage is used.

[CAN]

BLFFormat_Compatible_with_7_2_and_ealier = 0

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Channel the frame was sent or received.
mFlags	uint8_t	See 3.1.1
mDLC	uint8_t	Data length code of frame (number of valid data bytes, max. 8)
mID	uint32_t	Frame identifier.
mData[8]	uint8_t	CAN data bytes
mFrameLength	uint32_t	Message duration [in ns]. Not including 3 Interframe Space bit times and by Rx-messages also not including 1 End-Of-Frame bit time
mBitCount	uint8_t	Total number of bits of the message including EOF and Interframe Space [in bits]
mReserved1	uint8_t	Reserved, must be 0
mReserved2	uint16_t	Reserved, must be 0

3.4 VBLCANFDMessage64

Description: CAN FD data frame, or CAN data- or remote frame on a CAN FD channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_FD_MESSAGE_64`

Object available starting from CANoe/CANalyzer version 8.1

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>mHeader</td><td>VBLObjectHeader</td><td>Common header type. See 3.1.4.</td></tr>
<tr><td>mChannel</td><td>uint8_t</td><td>Channel the frame was sent or received.</td></tr>
<tr><td>mDLC</td><td>uint8_t</td><td>Data length code of the frame.
<table>
<tr><td>DLC</td><td colspan="2">Data length in bytes</td></tr>
<tr><td></td><td>CAN</td><td>CAN FD</td></tr>
<tr><td>0-8</td><td>0-8</td><td>0-8</td></tr>
<tr><td>9</td><td>8</td><td>12</td></tr>
<tr><td>10</td><td>8</td><td>16</td></tr>
<tr><td>11</td><td>8</td><td>20</td></tr>
<tr><td>12</td><td>8</td><td>24</td></tr>
<tr><td>13</td><td>8</td><td>32</td></tr>
<tr><td>14</td><td>8</td><td>48</td></tr>
<tr><td>15</td><td>8</td><td>64</td></tr>
</table></td></tr>
<tr><td>mValidDataBytes</td><td>uint8_t</td><td>Valid payload length of mData. The value is 0, if the message was an CAN remote frame.</td></tr>
<tr><td>mTxCount</td><td>uint8_t</td><td>Bits 0 – 3: Number of required transmission attempts
Bits 4 – 7: Max number of transmission attempts.</td></tr>
<tr><td>mID</td><td>uint32_t</td><td>Frame identifier.</td></tr>
<tr><td>mFrameLength</td><td>uint32_t</td><td>Message duration [in ns]. Not including 3 interframe-space bit times and by Rx-messages also not including one end-of-frame bit time</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>mFlags</td><td>uint32_t</td><td>
<table>
<tr><td>Bit#</td><td>Meaning</td></tr>
<tr><td>0 (0x0001)</td><td>Must be 0</td></tr>
<tr><td>1 (0x0002)</td><td>Reserved, for internal use</td></tr>
<tr><td>2 (0x0004)</td><td>1=NERR (1=single wire on low speed CAN)</td></tr>
<tr><td>3 (0x0008)</td><td>1=High voltage wake up</td></tr>
<tr><td>4 (0x0010)</td><td>1=Remote frame (only CAN)</td></tr>
<tr><td>5 (0x0020)</td><td>Reserved, must be 0</td></tr>
<tr><td>6 (0x0040)</td><td>1= Tx Acknowledge</td></tr>
<tr><td>7 (0x0080)</td><td>1= Tx Request</td></tr>
<tr><td>8 (0x0100)</td><td>Reserved, must be 0</td></tr>
<tr><td>9 (0x0200)</td><td>SRR (CAN FD)</td></tr>
<tr><td>10 (0x0400)</td><td>R0</td></tr>
<tr><td>11 (0x0800)</td><td>R1</td></tr>
<tr><td>12 (0x1000)</td><td>FDF
0: CAN frame
1: CAN FD frame</td></tr>
<tr><td>13 (0x2000)</td><td>BRS (CAN FD)</td></tr>
<tr><td>14 (0x4000)</td><td>ESI</td></tr>
<tr><td>15 (0x8000)</td><td>Internal use only</td></tr>
<tr><td>16 (0x10000)</td><td>Reserved, must be 0</td></tr>
<tr><td>17 (0x20000)</td><td>1= Frame is part of a burst</td></tr>
<tr><td>18 (0x40000)</td><td>Single shot mode: Frame could not be transmitted</td></tr>
<tr><td>19 (0x80000)</td><td>Single shot mode: If bit 18 is set to 1, then this bit reports the reason.
0 = arbitration lost,
1 = frame disturbed</td></tr>
<tr><td>20 (0x100000)</td><td>Reserved, for internal use</td></tr>
<tr><td>21 (0x200000)</td><td>Reserved for internal use</td></tr>
<tr><td>22-31</td><td>Reserved, must be 0</td></tr>
</table>
</td></tr>
<tr><td>mBtrCfgArb</td><td>uint32_t</td><td>CAN- or CAN-FD bit timing configuration for arbitration phase, may be 0, if not supported by hardware/driver

Bit 0-7: Quartz frequency in MHz
Bit 8-15: Prescaler
Bit 16-23: # of time quanta per bit
Bit 24-31: Sampling point in percent</td></tr>
<tr><td>mBtrCfgData</td><td>uint32_t</td><td>CAN-FD bit timing configuration for data phase, may be 0, if not supported by hardware/driver. See mBtrCfgArb.</td></tr>
</table>

Parameter	Type	Description
mTimeOffsetBrsNs	uint32_t	Time offset of the sampling point of BRS in nanoseconds
mTimeOffsetCRCDelNs	uint32_t	Time offset of the sampling point of CRC delimiter in nanoseconds
mBitCount	uint16_t	Bit count of the message, exclusive stuff bits.
mDir	uint8_t	Direction of the message
mExtDataOffset	uint8_t	Offset of the extended event data. Use the macros 'BLHasExtFrameData' and 'BLExtFrameDataPtr' to get a pointer to mExtFrameData. See example below.
mCRC	uint32_t	CRC of the message. For CAN FD ISO-frames the stuff count and additional flags are stored in the field: Bits: 0-20 — CRC; 21-26 — Reserved, must be 0; 27-29 — Stuff count field; 30 — Stuff count field parity; 31 — ISO format. If set to 1, then the message is in CAN FD ISO format, and the stuff count is valid.
mData[]	uint8_t	Data bytes of the message. The array size is set to the frame's data length stored in mValidDataBytes. The maximum value is 64.
mExtFrameData	struct VBLCANFDExtFrameData	See description below.

struct VBLCANFDExtFrameData

CANoe/CANalyzer version 8.5 and newer are supporting CAN FD ISO with extended bit timing configurations. The bit timing for the arbitration phase is stored in mBTRExtArb, the bit timing for the data phase is mBTRExtData.

The extended format is:

Bit 0 – 7: TSEG1-1
Bit 8 – 15: TSEG2-1
Bit 16 – 27: Prescaler
Bit 28 – 31: Quartz Frequency (enumeration). Supported values: 0: 16 MHz, 1: 32 MHz, 2: 80 MHz

Example:

```
VBLCANFDMessage64 blfEvt;
if (BLHasExtFrameData(&blfEvt)){
  unsigned int btrArb = BLExtFrameDataPtr(&blfEvt)->mBTRExtArb);
  unsigned int btrData = BLExtFrameDataPtr(&blfEvt)->mBTRExtData);
}
```

```
VBLCANFDErrorFrame64 blfEf;
if (BLHasExtFrameData(&blfEf)){
  unsigned int btrArb = BLExtFrameDataPtr(&blfEf)->mBTRExtArb);
  unsigned int btrData = BLExtFrameDataPtr(&blfEf)->mBTRExtData);
}
```

3.5 VBLCANErrorFrame

Description: CAN error frame received or transmitted on a CAN channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_ERROR`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Channel the frame was sent or received.
mLength	uint16_t	Length of error frame - can be left 0.

3.6 VBLCANErrorFrameExt

Description: Extended CAN error frame received or transmitted on a CAN channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_ERROR_EXT`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Channel the frame was sent or received.
mLength	uint16_t	Length of error frame, unused, may be 0.
mFlags	uint32_t	Defines what additional information is valid. Following values are possible: 1: SJA 1000 ECC is valid (member mECC) 2: Vector CAN Core Error Code is valid. 4: Vector CAN Core Error Position 8: Vector CAN Core Frame Length in ns

Parameter	Type	Description
mECC	uint8_t	Content of Philips SJA1000 Error Code Capture (ECC) register, or the Vector CAN-Core error register (see also mFlags). SJA1000-ECC See documentation of Philips SJA1000 CAN Controller. Vector CAN-Core Bit Meaning 0-5 0: Bit Error 1: Form Error 2: Stuff Error 3: Other Error 4: CRC Error 5: Ack-Del-Error 7: Ack-Error 6-7 0: RX-NAK-Error 1: TX-NAK-Error 2: RX-Error 3: TX-Error
mPosition [1,2]	uint8_t	Bit position of the error frame in the corrupted message.
mDLC [1, 2, 3]	uint8_t	Data length code of the corrupted message.
mReserved1	uint8_t	Reserved, must be 0.
mFrameLengthInNS [1,2]	uint32_t	Difference between the time stamp of the error frame and the start of frame in nanoseconds. Not all hardware interfaces are supporting this parameter.
mID [1, 2, 3]	uint32_t	Message ID of the corrupted message.

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>mFlagsExt [1]</td><td>uint16_t</td><td>Extended error flags.
<table>
<tr><td>Bit</td><td>Meaning</td></tr>
<tr><td>0-4</td><td>Segment (only SJA1000)</td></tr>
<tr><td>5</td><td>Direction, 1=RX</td></tr>
<tr><td>6-11</td><td>Error Code
0 Bit Error
1 Form Error
2 Stuff Error
3 Other Error
4 CRC Error [2]
5 ACK-DEL Error [2]
7 ACK Error [2]</td></tr>
<tr><td>12-13</td><td>Extended Direction [2]
0 RX NAK
1 TX NAK
2 RX
3 TX</td></tr>
<tr><td>14</td><td>1 = The error frame was send from the application</td></tr>
</table></td></tr>
<tr><td>mReserved2</td><td>uint16_t</td><td>Reserved, must be 0.</td></tr>
<tr><td>mData[8][1, 2, 3]</td><td>uint8_t</td><td>Message data.</td></tr>
</table>

Validity of the Parameters

1 Since CANoe/CANalyzer 7.5
2 Only valid for interfaces with Vector CAN-Core (CANcardXLe, VN7600, and others)
3 The validity of ID, DLC, and the data field depends on the type and position of the disturbance. Example: If the message is disturbed in the ID-field, then only the first bits of the ID may be valid, but not the DLC and the data field. The error position is not the position of the disturbance. Example: If the error position is located in the CRC-field, then the message may have been disturbed in any other field, and the erroneous CRC is the result of that disturbance.

3.7 VBLCANFDErrorFrame64

Description: CAN-FD error frame received or transmitted on a CAN-FD channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_FD_ERROR_64`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint8_t	Channel the frame was sent or received.
mDLC	uint8_t	Data length code of the corrupted message.
mValidDataBytes	uint8_t	Number of data bytes of the corrupted message. The value is 0, if the corrupted message was a CAN remote frame.
mECC	uint8_t	Content of Philips SJA1000 Error Code Capture register, or the Vector CAN-Core error register. See field mECC of VBLCANErrorFrameExt.

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>mFlags</td><td>uint16_t</td><td>Defines what additional information is valid. See field mFlags of VBLCANErrorFrameExt.</td></tr>
<tr><td>mErrorCodeExt</td><td>uint16_t</td><td>Extended error flags. See field mFlagsExt of VBLCANErrorFrameExt.</td></tr>
<tr><td>mExtFlags</td><td>uint16_t</td><td>CAN-FD specific flags.<table><tr><td>Bit</td><td>Meaning</td></tr><tr><td>0</td><td>0: FDF is 0 (CAN Error Frame)
1: FDF is 1 (CAN FD Error Frame)</td></tr><tr><td>1</td><td>0: BRS is 0
1: BRS is 1</td></tr><tr><td>2</td><td>0: ESI is 0
1: ESI is 1</td></tr><tr><td>3</td><td>0: Error in Arbitration Phase
1: Error in Data Phase</td></tr><tr><td>4</td><td>0: Error is on a CAN channel
1: Error is on a CAN FD channel</td></tr><tr><td>all others</td><td>reserved, must be set o 0</td></tr></table></td></tr>
<tr><td>mExtDataOffset</td><td>uint8_t</td><td>Offset of the extended event data. Use the macros ‘BLHasExtFrameData’ and ‘BLExtFrameDataPtr’ to get a pointer to mExtFrameData. See example in chapter 3.4.</td></tr>
<tr><td>reserved1</td><td>uint8_t</td><td>Reserved, must be 0.</td></tr>
<tr><td>mID</td><td>uint32_t</td><td>Message ID of the corrupted message.</td></tr>
<tr><td>mFrameLength</td><td>uint32_t</td><td>Difference between the time stamp of the error frame and the start of frame in nanoseconds. Not all hardware interfaces are supporting this parameter.</td></tr>
<tr><td>mBtrCfgArb</td><td>uint32_t</td><td>CAN-FD bit timing configuration for arbitration phase, may be 0, if not supported by hardware/driver

Bit 0-7: Quartz frequency
Bit 8-15: prescaler
Bit 16-23: # of time quanta of a bit
Bit 24-31: Sampling point in percent</td></tr>
<tr><td>mBtrCfgData</td><td>uint32_t</td><td>CAN-FD bit timing configuration for data phase, may be 0, if not supported by hardware/driver. See mBtrCfgArb.</td></tr>
<tr><td>mTimeOffsetBrsNs</td><td>uint32_t</td><td>Time offset of bit rate switch within BRS field in nanoseconds</td></tr>
<tr><td>mTimeOffsetCRCDelNs</td><td>uint32_t</td><td>Time offset of bit rate switch within CRC delimiter field in nanoseconds</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>mCRC</td><td>uint32_t</td><td>CRC of the message.

For CAN FD ISO-frames the stuff count and additional flags are stored in the field:
<table><tr><td>Bits</td><td>Meaning</td></tr><tr><td>0-20</td><td>CRC</td></tr><tr><td>21-26</td><td>Reserved, must be 0</td></tr><tr><td>27-29</td><td>Stuff count field</td></tr><tr><td>30</td><td>Stuff count field parity</td></tr><tr><td>31</td><td>ISO format. If set to 1, then the message is in CAN FD ISO format, and the stuff count is valid.</td></tr></table></td></tr>
<tr><td>mErrorPosition</td><td>uint16_t</td><td>Bit position of the error frame in the corrupted message.</td></tr>
<tr><td>mReserved2</td><td>uint16_t</td><td>Reserved, must be 0.</td></tr>
<tr><td>mData[]</td><td>uint8_t</td><td>Data bytes of the message. The array size is set to the frame’s data length stored in mValidDataBytes. The maximum value is 64.</td></tr>
<tr><td>mExtFrameData</td><td>struct VBLCANFDExtFrameData</td><td>See description in chapter 3.4..</td></tr>
</table>

Validity of the Parameters

For all fields the same restrictions as for VBLCANErrorFrameExt apply.

3.8 VBLCANOverloadFrame

Description: CAN overload frame received or transmitted on a CAN channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_OVERLOAD`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Channel the frame was sent or received.
mDummy	uint16_t	Reserved, must be 0

3.9 VBLCANDriverStatistic

Description: CAN driver statistic data for a CAN channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_STATISTIC`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	CAN channel the statistic data belongs to.

Parameter	Type	Description
mBusLoad	uint16_t	Busload in 1/100 percent (e.g. 100 means 1%)
mStandardDataFrames	uint32_t	Number of standard data frames sent on that channel.
mExtendedDataFrames	uint32_t	Number of extended data frames sent on that channel.
mStandardRemoteFrames	uint32_t	Number of remote data frames sent on that channel.
mExtendedRemoteFrames	uint32_t	Number of extended remote data frames sent on that channel.
mErrorFrames	uint32_t	Number of error frames sent on that channel
mOverloadFrames	uint32_t	Number of overload frames sent on that channel.

3.10 VBLCANDriverError

Description: CAN driver error information for transceiver of a CAN channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_DRIVER_ERROR`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	CAN channel the driver error information belongs to.
mTXErrors	uint8_t	Number of transmit errors that occurred in CAN controller for that channel.
mRXErrors	uint8_t	Number of receive errors that occurred in CAN controller for that channel.
mErrorCode	uint32_t	See 3.1.2

3.11 VBLCANDriverErrorExt

Description: Extended CAN driver error information for transceiver of a CAN channel. **This object is currently not used in CANoe / CANalyzer.**

Corresponding object type: `BL_OBJ_TYPE_CAN_DRIVER_ERROR_EXT`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	CAN channel the driver error information belongs to.

Parameter	Type	Description
mTXErrors	uint8_t	Number of transmit errors that occurred in CAN controller for that channel.
mRXErrors	uint8_t	Number of receive errors that occurred in CAN controller for that channel.
mErrorCode	uint32_t	See 3.1.2.
mFlags	uint32_t	To be defined.
mState	uint8_t	To be defined.
mReserved1	uint8_t	Reserved, must be 0
mReserved2	uint16_t	Reserved, must be 0
mReserved3[4]	uint32_t	Reserved, must be 0

3.12 VBLCANDriverHwSync

Description: Event that occurs when hardware sync is executed.

Corresponding object type: `BL_OBJ_TYPE_CAN_DRIVER_SYNC`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Application channel
mFlags	uint8_t	The following values are possible: 1: sync was sent from this channel (`BL_HWSYNC_FLAGS_TX`) 2: external sync received (`BL_HWSYNC_FLAGS_RX`) 4: sync received but generated from this hardware (`BL_HWSYNC_FLAGS_RX_THIS`)
mDummy	uint8_t	Reserved, must be 0

3.13 VBLCANSettingsChanged

Description: Event that occurs when a reset or a bit timing change occurs on a CAN channel.

Corresponding object type: `BL_OBJ_TYPE_CAN_DRIVER_SYNC`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Application channel

Parameter	Type	Description
mChangeType	uint8_t	The following values are possible: -1: Invalid Change Type 0: Reset event 1: Bit timing changed
mBitTimings	struct VBLCANFDExtFrameData	See description in chapter 3.4. If CAN channel mBTRExtData is 0.

3.14 VBLEnvironmentVariable

Description: Environment variable that can be used with CANoe.

Corresponding object types:

- BL_OBJ_TYPE_ENV_INTEGER
- BL_OBJ_TYPE_ENV_double
- BL_OBJ_TYPE_ENV_STRING
- BL_OBJ_TYPE_ENV_DATA

Note that the object type depends on the type of the environment variable's data. E.g. if you want to save an environment variable with data of type `double`, set the object type to `BL_OBJ_TYPE_double`.

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mNameLength	uint32_t	Length of the name of the environment variable (without terminating 0)
mDataLength	uint32_t	Length of the data of the environment variable in bytes.
mName	BL_LPSTR	Name of the environment variable.
mData	BL_LPBYTE	Data value of the environment variable.

Note:
The size of a VBLEnvironmentVariable object depends on the length of the name and the length of the data. To set the size correctly you have to add the length of the name and the length of the data to the object size:

```
VBLEnvironmentVariable envVar;

envVar.mHeader.mBase.mObjectSize = sizeof(VBLEnvironmentVariable) +
envVar.mNameLength + envVar.mDataLength;
```

The following piece of code shows how to save different environment variables to a BLF file.

```
uint8_t bytes[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
char* string = "Hello world";
double doubleData = 4.2;
```

```
BLHANDLE hFile;
// open / create the BLF file
// ...
VBLEnvironmentVariable envvar;
envvar.mName = "envvar1";
envvar.mNameLength = strlen("envvar1");

envvar.mHeader.mBase.mObjectType = BL_OBJ_TYPE_ENV_DATA;
envvar.mData = bytes;
envvar.mDataLength = sizeof( bytes);
envvar.mHeader.mBase.mObjectSize = sizeof( VBLEnvironmentVariable) +
  envvar.mDataLength + envvar.mNameLength;
BLWriteObject( hFile, &envvar.mHeader.mBase);

envvar.mHeader.mBase.mObjectType = BL_OBJ_TYPE_ENV_double;
envvar.mData = (BL_LPBYTE) &doubleData;
envvar.mDataLength = sizeof( doubleData);
envvar.mHeader.mBase.mObjectSize = sizeof( VBLEnvironmentVariable) +
  envvar.mDataLength + envvar.mNameLength;
BLWriteObject( hFile, &envvar.mHeader.mBase);

envvar.mHeader.mBase.mObjectType = BL_OBJ_TYPE_ENV_STRING;
envvar.mData = (BL_LPBYTE) string;
envvar.mDataLength = strlen( string);
envvar.mHeader.mBase.mObjectSize = sizeof( VBLEnvironmentVariable) +
  envvar.mDataLength + envvar.mNameLength;
BLWriteObject( hFile, &envvar.mHeader.mBase);
```

The following piece of code shows you how to read in and use environment variables:

```
BLHANDLE hFile;
// open the BLF file
// ...
VBLObjectHeaderBase base;
VBLEnvironmentVariable envvar;
uint8_t* data = NULL;
double doubleData = 0.0;
char* stringdata = NULL;

while( BLPeekObject( hFile, &base))
{
  switch( base.mObjectType)
  {
  case BL_OBJ_TYPE_ENV_DATA:
    envvar.mHeader.mBase =  base;
    BLReadObject( hFile, &envvar.mHeader.mBase);
    data = (uint8_t*) malloc( envvar.mDataLength);
    memcpy( data, envvar.mData, envvar.mDataLength);
    BLFreeObject( hFile, &envvar.mHeader.mBase);
    break;
  case BL_OBJ_TYPE_ENV_double:
    envvar.mHeader.mBase = base;
    BLReadObject( hFile, &envvar.mHeader.mBase);
    memcpy( &doubleData, envvar.mData, sizeof( doubleData));
    BLFreeObject( hFile, &envvar.mHeader.mBase);
    break;
  case BL_OBJ_TYPE_ENV_STRING:
    envvar.mHeader.mBase = base;
    BLReadObject( hFile, &envvar.mHeader.mBase);
    stringdata = (char*) malloc( envvar.mDataLength+1);
    memcpy( stringdata, envvar.mData, envvar.mDataLength);
    stringdata[envvar.mDataLength] = 0;
    BLFreeObject( hFile, &envvar.mHeader.mBase);
    break;
  default:
```

```
      BLSkipObject( hFile, &base);
      break;
    }
  }
```

3.15 VBLSystemVariable

Description: System variable that can be used with CANoe.

Corresponding object type: `BL_OBJ_TYPE_SYS_VARIABLE`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mType	uint32_t	Type of system variable. Following values are possible: 1: double (BL_SYSVAR_TYPE_double) 2: LONG (BL_SYSVAR_TYPE_LONG) 3: STRING (BL_SYSVAR_TYPE_STRING) 4: Array of double (BL_SYSVAR_TYPE_doubleARRAY) 5 Array of LONG (BL_SYSVAR_TYPE_LONGARRAY) 6: LONGLONG (BL_SYSVAR_TYPE_LONGLONG) 7: Array of uint8_t (BL_SYSVAR_TYPE_uint8_tARRAY)
mRepresentation	uint32_t	If mType is LONG or LONGLONG: 0 if the data value is signed, 1 if the data value is unsigned. For other types, undefined and must be 0.
mReserved[2]	uint32_t	Reserved, must be 0.
mNameLength	uint32_t	Length of the name of the system variable (without terminating 0)
mDataLength	uint32_t	Length of the data of the environment variable in bytes.
mName	BL_LPSTR	Name of the system variable.
mData	BL_LPBYTE	Data value of the system variable.

Note:
The size of a VBLSystemVariable object depends on the length of the name and the length of the data. To set the size correctly you have to add the length of the name and the length of the data to the object size:

```
VBLSystemVariable sysVar;
sysVar.mHeader.mBase.mObjectSize = sizeof(VBLSystemVariable) +
sysVar.mNameLength + sysVar.mDataLength;
```

The following piece of code shows how to save different system variables to a BLF file.

```
uint8_t bytes[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
char* string = "Hello world";
double doubleData = 4.2;
```

```
BLHANDLE hFile;
// open / create the BLF file
// ...
VBLSystemVariable sysVar;
sysVar.mName = "sysVar1";
sysVar.mNameLength = strlen( "sysVar1");

sysVar.mType = BL_SYSVAR_TYPE_uint8_tARRAY;
sysVar.mData = bytes;
sysVar.mDataLength = sizeof( bytes);
sysVar.mHeader.mBase.mObjectSize = sizeof( VBLSystemVariable) +
  sysVar.mDataLength + sysVar.mNameLength;
BLWriteObject( hFile, &sysVar.mHeader.mBase);

sysVar.mType = BL_SYSVAR_TYPE_double;
sysVar.mData = (BL_LPBYTE) &doubleData;
sysVar.mDataLength = sizeof( doubleData);
sysVar.mHeader.mBase.mObjectSize = sizeof( VBLSystemVariable) +
  sysVar.mDataLength + sysVar.mNameLength;
BLWriteObject( hFile, &sysVar.mHeader.mBase);

sysVar.mType = BL_SYSVAR_TYPE_STRING;
sysVar.mData = (BL_LPBYTE) string;
sysVar.mDataLength = strlen( string);
sysVar.mHeader.mBase.mObjectSize = sizeof( VBLSystemVariable) +
  sysVar.mDataLength + sysVar.mNameLength;
BLWriteObject( hFile, &sysVar.mHeader.mBase);
```

The following piece of code shows you how to read in and use environment variables:

```
BLHANDLE hFile;
// open the BLF file
// ...
VBLObjectHeaderBase base;
VBLSystemVariable sysVar;
uint8_t* sysvardata = NULL;
double sysDouble = 0.0;
char* sysstring = NULL;

while( BLPeekObject( hFile, &base))
{
  switch( base.mObjectType)
  {
  case BL_OBJ_TYPE_SYS_VARIABLE:
    sysVar.mHeader.mBase = base;
    BLReadObject( hFile, &sysVar.mHeader.mBase);
    switch(sysVar.mType)
    {
    case BL_SYSVAR_TYPE_uint8_tARRAY:
      sysvardata = (uint8_t*) malloc( sysvar.mDataLength);
      memcpy( sysvardata, sysVar.mData, sysvar.mDataLength);
      break;
    case BL_SYSVAR_TYPE_double:
      memcpy( &sysDouble, sysVar.mData, sizeof( sysDouble));
      break;
    case BL_SYSVAR_TYPE_STRING:
      sysstring = (char*) malloc(sysVar.mDataLength+1);
      memcpy( sysstring, sysVar.mData, sysVar.mDataLength);
      sysstring[sysVar.mDataLength] = 0;
      break;
    default:
      break;
    }
    BLFreeObject( hFile, & sysVar.mHeader.mBase);
    break;
```

```
    default:
      BLSkipObject( hFile, &base);
      break;
    }
  }
```

3.16 VBLGPSEvent

Description: GPS event.

Corresponding object type: `BL_OBJ_TYPE_GPS_EVENT`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mFlags	uint32_t	Not used, must be 0.
mChannel	uint16_t	GPS channel the GPS event was sent.
mReserved	uint16_t	Reserved, must be 0,
mLatitude	double	Latitude, possible values reach from -180 to 180. Negative values are western hemisphere, positive values are eastern hemisphere.
mLongitude	double	Longitude, possible values reach from -90 to 90. Negative values are Southern hemisphere, positive values are northern hemisphere.
mAltitude	double	Altitude in meters, measured above sea line.
mSpeed	double	Current vehicle speed in km/h.
mCourse	double	Current driving course, possible values reach from -180 to 180. A value of 0 means driving north, 90 means driving east, -90 means driving west, -180 and 180 mean driving south.

3.17 VBLWlanFrame

Description: WLAN frame.

Corresponding object type: `BL_OBJ_TYPE_WLAN_FRAME`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	The channel of the frame.
mFlags	uint16_t	Bit 0 – Genuine MAC Header Bit 1 – Correct Frame Control Format
mDir	uint8_t	Direction flag: 0=Rx, 1=Tx 2=TxRq
mRadioChannel	uint8_t	Channel number of the radio frequency, i.e 180 or 176

Parameter	Type	Description
mSignalStrength	int16_t	Signal strength in [dBm]
mSignalQuality	uint16_t	Signal quality
mFrameLength	uint16_t	Length of WLAN data in bytes. Max. 2342 Bytes.
mFrameData	uint8_t*	WLAN frame data. Data starts with WLAN header.

Note:
The size of a VBLWlanFrame object depends on the length of the frame data. To set the size correctly you have to add the payload data length to the object size:

```
VBLWlanFrame wlanFrame;

wlanFrame.mHeader.mBase.mObjectSize = sizeof(VBLWlanFrame) +
wlanFrame.mFrameLength;
```

3.18 VBLWlanStatistic

Description: WLAN statistic event.

Corresponding object type: `BL_OBJ_TYPE_WLAN_STATISTIC`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	The channel of the frame.
mFlags	uint16_t	Bit 0 – Valid Rx/Tx counter Bit 1 – Valid error counter (collisions and errors)
mRxPacketCount	uint32_t	Number of Rx packets since last statistic event.
mRxByteCount	uint32_t	Number of Rx bytes since last statistic event.
mTxPacketCount	uint32_t	Number of Tx packets since last statistic event.
mTxByteCount	uint32_t	Number of Tx packets since last statistic event.
mCollisionCount	uint32_t	Number of collisions since last statistic event.
mErrorCount	uint32_t	Number of errors since last statistic event.

3.19 VBLTriggerCondition

Description: Trigger Condition event.

Corresponding object type: `BL_OBJ_TYPE_TRIGGER_CONDITION`

Parameter	Type	Description
mHeader	VBLVarObjectHeader	Common header type. See 3.1.6.
mStatic		

Parameter	Type	Description
mState	uint32_t	BL_TC_STATUS_UNKNOWN BL_TC_STATUS_START BL_TC_STATUS_STOP BL_TC_STATUS_STARTSTOP
mTriggerBlockNameLength	uint32_t	Length of mTriggerBlockName
mTriggerConditionLength	uint32_t	Length of mTriggerCondition
mDynamic		
mTriggerBlockName	BL_LPSTR	Trigger Block Name that generated the trigger condition
mTriggerCondition	BL_LPSTR	Trigger Condition description

3.20 VBLAppTrigger

Description: Application defined trigger to be saved in BLF log file (**currently not used in CANoe / CANalyzer**).

Corresponding object type: `BL_OBJ_TYPE_APP_TRIGGER`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mPreTriggerTime	uint32_t	Pre trigger time.
mPostTriggerTime	uint32_t	Post trigger time.
mChannel	uint16_t	Trigger that channel belongs to.
mFlags	uint16_t	0: single trigger type 1: start logging trigger type 2: stop logging trigger type
mAppSecific2	uint32_t	Reserved.

3.21 VBLAppText

Description: Application defined text to be saved in BLF log file (**currently not used in CANoe/CANalyzer**).

Corresponding object type: `BL_OBJ_TYPE_APP_TEXT`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>mSource</td><td>uint32_t</td><td>Defines the source/semantic of the text. Actually, two different values are defined:

0: BL_APPTEXT_MEASUREMENTCOMMENT
mReserved is not used
mText contains a measurement comment

1: BL_APPTEXT_DBCHANNELINFO
mReserved contains channel information. The following table shows how the 4 bytes are used
<table><tr><td>Bit 0-7</td><td>Version of the data</td></tr><tr><td>Bit 8-15</td><td>Channel number</td></tr><tr><td>Bit 15-23</td><td>Bus type of the channel. One of the following values:
BL_BUSTYPE_CAN 1
BL_BUSTYPE_LIN 5
BL_BUSTYPE_MOST 6
BL_BUSTYPE_FLEXRAY 7
BL_BUSTYPE_J1708 9
BL_BUSTYPE_ETHERNET 10
BL_BUSTYPE_WLAN 13
BL_BUSTYPE_AFDX 14</td></tr><tr><td>Bit 24</td><td>Flag, that determines, if channel is a CAN-FD channel</td></tr><tr><td>Bit 25-31</td><td>Currently unused.</td></tr></table>
mText contains database information for the specified channel. Each database is defined by the database path and the cluster name (if available). The single databases and the cluster name are separated by a semicolon. Example:

<Path1>;<ClusterName1>;<Path2>;<ClusterName2>;…

If for a database there’s no cluster name available, an empty string is written as cluster name.

2: BL_APPTEXT_ATTACHMENT

mReserved is not used

mText contains are relative or absolute path, or a URL to an associated file. Optionally, a MIME-type can be added after a separating semicolon. Example:

./CAN1.dbc;database/dbc</td></tr>
<tr><td>mReserved</td><td>uint32_t</td><td>Depends on mSource.</td></tr>
<tr><td>mTextLength</td><td>uint32_t</td><td>Length of mText without ending 0.</td></tr>
<tr><td>mText</td><td>BL_LPSTR</td><td>Text to be saved to log file.</td></tr>
</table>

Note:
The size of a VBLAppText object depends on the length of the text. To set the size correctly you have to add the text length to the object size:

```
VBLAppText appText;
appText.mHeader.mBase.mObjectSize = sizeof(VBLAppText) +
appText.mTextLength;
```

3.22 VBLEventComment

Description: Comment of an event. The comment can be set in Trace Window.

Corresponding object type: `BL_OBJ_TYPE_EVENT_COMMENT`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mCommentedEventType	uint32_t	Type of the commented event
mTextLength	uint32_t	Length of mText without ending 0.
mText	BL_LPSTR	Comment text.

Note:
The size of a VBLEventComment object depends on the length of the text. To set the size correctly you have to add the text length to the object size:

```
VBLEventComment comment;

comment.mHeader.mBase.mObjectSize = sizeof(VBLEventComment) +
comment.mTextLength;
```

3.23 VBLGlobalMarker

Description: Global Marker assigned to another event or to a time stamp.

Corresponding object type: `BL_OBJ_TYPE_GLOBAL_MARKER`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mCommentedEventType	uint32_t	Type of the commented event
mForegroundColor	uint32_t	Foreground color of the marker group
mBackgroundColor	uint32_t	Background color of the marker group
mIsRelocatable	uint8_t	Defines whether a marker can be relocated
mGroupNameLength	uint32_t	Length of mGroupName without ending 0.
mMarkerNameLength	uint32_t	Length of mMarkerName without ending 0.
mDescriptionLength	uint32_t	Length of mDescription without ending 0.
mGroupName	BL_LPSTR	Group name.
mMarkerName	BL_LPSTR	Marker.
mDescription	BL_LPSTR	Description text.

Note:
The size of a VBLGlobalMarker object depends on the length of the text (group name, marker name, description). To set the size correctly you have to add the text length to the object size:

```
VBLEventComment comment;
comment.mHeader.mBase.mObjectSize = sizeof( VBLEventComment) +
mGroupNameLength + mMarkerNameLength + mDescriptionLength
```

3.24 VBLAfdxFrame

Description: AFDX frame.

Corresponding object type: `BL_OBJ_TYPE_AFDX_FRAME`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mSourceAddress[6]	uint8_t	Ethernet (MAC) address of source computer (network byte order).
mChannel	uint16_t	The channel of the frame.
mDestinationAddress[6]	uint8_t	Ethernet (MAC) address of target computer (network byte order).
mDir	uint16_t	Direction flag: 0=Rx, 1=Tx, 2=TxRq
mType	uint16_t	EtherType which indicates protocol for Ethernet payload data See Ethernet standard specification for valid values.
mTPID	uint16_t	TPID when VLAN tag valid, zero when no VLAN. See Ethernet standard specification.
mTCI	uint16_t	TCI when VLAN tag valid, zero when no VLAN. See Ethernet standard specification.
mEthChannel	uint8_t	Channel number of the underlying Ethernet interface, where the frame originated from.

Parameter	Type	Description
mAfdxFlags	uint16_t	Status- and error flags as: Bit 0 – Frame from line-B Bit 1 – Frame is redundant Bit 2 – Frame is a fragment of a message Bit 3 – Packet is a SAP message or part of it Bit 4 – Frame occurred on wrong line (A/B) Bit 5 – Frame is not a valid AFDX frame Bit 6 – AFDX-sequenceNo is invalid Bit 7 – Redundancy error encountered Bit 8 – Fragmentation / reassembly error Bit 9 – Violation of a higher protocol Bit 10 – Packet has been reassembled. Bit 11 – Frame has been checked by redundancy manager Bit 12 – a constant or value violates AFDX recommendation Bit 13 – Frame size does not match expected size from database Bit 14 – Packet is not defined in database Bit 15 – Frame has been checked by integrity manager
mBAGusec	uint32_t	Time period since last received frame on this virtual link in micro-seconds
mPayLoadLength	uint16_t	Length of Ethernet payload data in bytes. Max. 1500 Bytes (without Ethernet header) as per AFDX-spec; raw Ethernet may have 1582 Bytes
mPayLoad	uint8_t*	Ethernet payload data (without Ethernet header)

Note:
The size of a VBLAfdxFrame object depends on the length of the payload data. To set the size correctly you have to add the payload data length to the object size:

```
VBLAfdxFrame afdxFrame;

afdxFrame.mHeader.mBase.mObjectSize = sizeof(VBLAfdxFrame) +
afdxFrame.mPayLoadLength;
```

It follows the same schema as a VBLEthernetFrame, just with additional elements.

3.25 VBLAfdxStatistic

Description: AFDX statistic event per virtual link.

Corresponding object type: `BL_OBJ_TYPE_AFDX_STATISTIC`

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	The channel of the frame.

Parameter	Type	Description
mFlags	uint16_t	Bit 0 – Valid Rx/Tx counter Bit 1 – Valid error counter Bit 2 – Valid VirtualLink ID
mRxPacketCount	uint32_t	Number of Rx packets since last statistic event.
mRxByteCount	uint32_t	Number of Rx bytes since last statistic event.
mTxPacketCount	uint32_t	Number of Tx packets since last statistic event.
mTxByteCount	uint32_t	Number of Tx packets since last statistic event.
mCollisionCount	uint32_t	Number of collisions since last statistic event.
mErrorCount	uint32_t	Number of errors since last statistic event.
mStatDroppedRedundantPacketCount	uint32_t	Number of dropped packet due to redundancy check since last statistic event.
mStatDroppedRedundantErrorCount	uint32_t	Number of errors found at redundancy check since last statistic event.
mStatDroppedIntegrityErrorCount	uint32_t	Number of errors found at integrity check since last statistic event.
mStatAvrgPeriodMsec	uint32_t	Average period of frames on this VL in [msec].
mStatAvrgJitterMysec	uint32_t	Average jitter of the time period of frames on this VL in [mysec].
mVLId	uint32_t	Unique ID assigned to this VL.
mStatDuration	uint32_t	Time period covered by this event in [msec].

3.26 VBLAfdxBusStatistic

Description: AFDX bus statistic event per channel and line (adapter).

Corresponding object type: `BL_OBJ_TYPE_AFDX_BUS_STATISTIC`

Object available starting from CANoe/CANalyzer version 8.2

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.

Parameter	Type	Description
mChannel	uint16_t	The channel of the frame.
mFlags	uint16_t	Bit 0 – channel is configured Bit 1 – HW related counters valid (mStatRxPacketCountHW, mStatTxPacketCountHW, mStatRxErrorCountHW, mStatTxErrorCountHW, mStatRxBytesHW, mStatTxBytesHW) Bit 2 – CANwin related counters are valid (mStatRxPacketCount, mStatTxPacketCount) Bit 3 – link-related info is valid (mLinkStatus, mLinkSpeed, mLinkLost) Bit 4 – invalid packet counter is valid Bit 5 – lost packet counter is valid Bit 6 – dropped packet counter is valid Bit 7 – byte counters are based on CANwin packets, not HW
mStatDuration	uint32_t	Data collection period (epoche) [mysec].
mStatRxPacketCountHW	uint32_t	Read frames taken from hardware.
mStatTxPacketCountHW	uint32_t	Sent frames taken from hardware.
mStatRxErrorCountHW	uint32_t	Number of erronous Rx frames detected by hardware in epoche.
mStatTxErrorCountHW	uint32_t	Number of erronous Tx frames detected by hardware in epoche.
mStatRxBytesHW	uint32_t	Bytes received by hardware in epoche.
mStatTxBytesHW	uint32_t	Bytes sent by hardware in epoche.
mStatRxPacketCount	uint32_t	Received frames within CANwin.
mStatTxPacketCount	uint32_t	Sent frames from CANwin.
mStatDroppedPacketCount	uint32_t	Number of frames dropped actively by CANwin.
mStatInvalidPacketCount	uint32_t	Number of frames detected in CANwin with incompatible Eth-header.
mStatLostPacketCount	uint32_t	Number of frames lost by CANwin due to queue overflow etc.
mLine	uint8_t	LineA (0) or LineB (1)
mLinkStatus	uint8_t	Status of adapter as per Eth-Status.

Parameter	Type	Description
mLinkSpeed	uint16_t	Link speed : 0 : 10Mbps 1 : 100Mbps 2 : 1000Mbps
mLinkLost	uint16_t	Number of link-losses during epoche.

3.27 VBLAfdxLineStatus

Description: AFDX link status event per adapter, used within VBLAfdxStatus, not directly

Corresponding object type: `BL_OBJ_TYPE_AFDX_STATUS`

Parameter	Type	Description
mFlags	uint16_t	Valid fields (as ETH): Bit 0 – Link status Bit 1 – Bitrate Bit 2 – Ethernet Phy Bit 3 – Duplex Bit 4 – MDI Type Bit 5 – Connector Bit 6 – Clock Mode Bit 7 – BroadR-Reach Pair
mLinkStatus	uint8_t	Link Status: 0: Unknown 1: Down 2: Up 3 Negotiate 4: Link error
mEthernetPhy	uint8_t	Ethernet Phy: 0: Unknown 1: IEEE 802.3 2: BroadR-Reach
mDuplex	uint8_t	Duplex: 0: Unknown 1: Half Duplex 2: Full Duplex.
mMdi	uint8_t	0: Unknown 1: Direct 2: Crossover.
mConnector	uint8_t	0: Unknown 1: RJ45 2: D-Sub.
mClockMode	uint8_t	0: Unknown 1: Master 2: Slave.

Parameter	Type	Description
mPairs	uint8_t	0: Unknown 1: BR 1-pair 2: BR 2-pair 3: BR 4-pair.
mReserved	uint8_t	unused.
mBitrate	uint32_t	Bitrate in [kbit/s].

3.28 VBLAfdxStatus

Description: AFDX link status event per channel

Corresponding object type: `BL_OBJ_TYPE_AFDX_STATUS`

Object available starting from CANoe/CANalyzer version 8.2

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	The channel of the frame.
mStatusA	VBLAfdxLineStatus	Status of adapter lineA.
mStatusB	VBLAfdxLineStatus	Status of adapter lineB.

3.29 VBLAfdxErrorEvent

Description: AFDX general error event for asynchronous errors, i.e. not related directly to frame events.

Corresponding object type: `BL_OBJ_TYPE_AFDX_ERROR_EVENT`

Object available starting from CANoe/CANalyzer version 8.5

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Related channel or 0 if independent from channels
mErrorLevel	uint16_t	0: error 1: warning 2: informational.
mSourceIdentifier	uint32_t	Internal identifier of error source.
mErrorText	char[512]	Textual description of the error.
mErrorAttributes	char[512]	Pairs of token and description for further classification. The tokens depend on the error source.

3.30 VBLA429Message

Description: A429 word with additional attributes.

Corresponding object type: `BL_OBJ_TYPE_A429_MESSAGE`

Object available starting from CANoe/CANalyzer version 8.5

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mA429Data	uint8_t[4]	Complete raw A429 traffic bytes
mChannel	uint16_t	Related channel or 0 if independent from channels
mDir	uint8_t	Direction flag: 0=Rx, 1=Tx.
mBitrate	uint32_t	Bitrate in bits per sec. For Rx this is a measurement value, for Tx it's the requested send bitrate.
mErrReason	uint32_t	Bit-coded error number representing the error reason: 0 – No Error 1 – Gap Error 2 – Parity Error 4 – Bitrate too low 8 – Bitrate too High 16 – Frame Format Error 32 - Coding NRZ Error
mErrPosition	uint16_t	Bit position of error. Valid range is between bit position 0 and 31.
mFrameGap	uint32_t	Distance between two frames in ns
mFrameLength	uint32_t	Time between start of frame and end of frame in ns
mMsgCtrl	uint16_t	Message control for Tx messages. Denotes values for controlling the transmission of a message 0 – On Request 1 – Cyclic 2 – Cyclic Of
mCycleTime	uint32_t	Cycle Time in µs for Message Control parameter
mError	uint32_t	reserved
mBitLenOfLastBit	uint32_t	Bit length of the last bit I case of Rx-Error in ns

3.31 VBLA429ErrorEvent

Description: A429 general error event for asynchronous errors, i.e. not related directly to frame events.

Corresponding object type: `BL_OBJ_TYPE_A429_ERROR`

Object available starting from CANoe/CANalyzer version 8.5

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Related channel or 0 if independent from channels
mErrorType	uint16_t	0: error 1: warning 2: informational.
mSourceIdentifier	uint32_t	Internal identifier of error source.
mErrReason	uint32_t	Internal error reason number.
mErrorText	char[512]	Textual description of the error.
mErrorAttributes	char[512]	Pairs of token and descriptions for further classification. The tokens depend on the error source.

3.32 VBLA429Status

Description: A429 channel status event, describing the so-called channel parameter settings.

Corresponding object type: `BL_OBJ_TYPE_A429_STATUS`

Object available starting from CANoe/CANalyzer version 8.5

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Channel described
mDir	uint8_t	Channel direction type: 0: Rx, l: Tx.
mParity	uint16_t	Protocol parity to apply: 0: use hardware default, 1: disabled, 2: odd, 3: even, 4: mark, 5: space
mMinGap	uint32_t	Minimum gap to preserve between 2 messages in 1/8 bit time.
mBitrate	uint32_t	Tx channel (transmit) bitrate in Hz. Not relevant for Rx channels.

Parameter	Type	Description
mMinBitrate	uint32_t	Minimum allowed bitrate for Rx channels in Hz.
mMaxBitrate	uint32_t	Maximum allowed bitrate for Rx channels in Hz

3.33 VBLA429BusStatistic

Description: A429 bus statistic events.

Corresponding object type: `BL_OBJ_TYPE_A429_BUS_STATISTIC`

Object available starting from CANoe/CANalyzer version 8.5

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mChannel	uint16_t	Channel described
mDir	uint8_t	Channel direction type: 0: Rx, l: Tx.
mBusload	uint32_t	Bus load in 0.01 %.
mDataTotal	uint32_t	Counts of valid A429words during epoche.
mErrorTotal	uint32_t	Counts of invalid A429words during epoche, i.e. A429word events with error bits set.
mBitrate	uint32_t	Average bitrate during epoche.
mParityErrors	uint16_t	Parity error counts during epoche.
mBitrateErrors	uint16_t	Bitrate error counts during epoche.
mGapErrors	uint16_t	Gap error counts during epoche.
mLineErrors	uint16_t	Line error counts during epoche.
mFormatErrors	uint16_t	Format error counts during epoche
mDutyFactorErrors	uint16_t	Duty factor error counts during epoche
mWordLenErrors	uint16_t	Word length error count during epoche
mCodingErrors	uint16_t	Coding error count during epoche
mIdleErrors	uint16_t	Idle error count during epoche

Parameter	Type	Description
mLevelErrors	uint16_t	Level error count during epoche
mLabelCount	uint16_t[256]	Counts of valid A429words per label (index = label).

3.34 VBLTestStructure

Description: Events produced during execution of Test Modules or Test Configurations in CANoe. For each start or end of a structural element in the test, e.g. TestCase or TestGroup, a matching event is produced.

Corresponding object type: `BL_OBJ_TYPE_TEST_STRUCTURE`

Object available starting from CANoe/CANalyzer version 8.5 SP3

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mExecutingObjectIdentity	uint32_t	Unique ID for the executing Test Configuration or Test Module. This ID can be used to correlate all VBLTestStructure events during one measurement; it's not persistent across measurements.

Parameter	Type	Description
mType	uint16_t	Type of structure element in the test. The following values are defined: 1 Test Module (BL_TESTSTRUCT_TYPE_TM_TESTMODULE) 2 Test Group (in Test Module) (BL_TESTSTRUCT_TYPE_TM_TESTGROUP) 3 Test Case (in Test Module) (BL_TESTSTRUCT_TYPE_TM_TESTCASE) 8 Test Configuration (BL_TESTSTRUCT_TYPE_TESTCONFIGURATION) 9 Test Unit (BL_TESTSTRUCT_TYPE_TESTUNIT) 10 Test Group (in Test Unit) (BL_TESTSTRUCT_TYPE_TESTGROUP) 11 Test Fixture (BL_TESTSTRUCT_TYPE_TESTFIXTURE) 12 Test Sequence (BL_TESTSTRUCT_TYPE_TESTSEQUENCE) 13 Test Sequence List (BL_TESTSTRUCT_TYPE_TESTSEQUENCELIST) 14 Test Case (in Test Unit) (BL_TESTSTRUCT_TYPE_TESTCASE) 15 Test Case List (BL_TESTSTRUCT_TYPE_TESTCASELIST)
mUniqueNo	uint32_t	Unique ID for the structure element (Test Case etc.), can be used to correlate events belonging to the same element and to disambiguate between elements with the same mName. This value is not persistent across measurements. (Currently not supported for Elements not visible in the CANoe GUI, e.g. Test Cases inside a Test Sequence, these always have a value of 0xFFFFFFFF)
mAction	uint16_t	Defines if this event indicates the start, end of, or abort information for the structural element: 1 Start (BL_TESTSTRUCT_ACTION_BEGIN) 2 End (BL_TESTSTRUCT_ACTION_END) 3 Start (BL_TESTSTRUCT_ACTION_ABORT) Abort information is additional if test was aborted e.g. due to verdict impact setting or due to user stop; it is always followed by an end event.

Parameter	Type	Description
mResult	uint16_t	For “END” events, the overall result (verdict) of the structural element. The following values are defined: 0 Undefined (BL_TESTSTRUCT_VERDICT_UNDEFINED) 1 None (BL_TESTSTRUCT_VERDICT_NONE) 2 Passed (BL_TESTSTRUCT_VERDICT_PASSED) 3 Inconclusive (BL_TESTSTRUCT_VERDICT_INCONCLUSIVE) 4 Failed (BL_TESTSTRUCT_VERDICT_FAILED) 5 Error in test system (BL_TESTSTRUCT_VERDICT_ERRORINTESTSYSTEM)
mExecutingObjectNameLength	uint32_t	Length of mExecutingObjectName in wide characters without terminating null character.
mNameLength	uint32_t	Length of mName in wide characters without terminating null character.
mTextLength	uint32_t	Length of mText in wide characters without terminating null character.
mExecutingObjectName	BL_LPWSTR	Name of the executing Test Configuration or Test Module.
mName	BL_LPWSTR	Name of the structure element.
mText	BL_LPWSTR	Complete informational text of the event as shown in the CANoe Trace Window or in ASC-Log.

Note:
The size of a VBLTestStructure object depends on the length of the wide character strings (executing object name, element name, element info). To calculate the size correctly you have to add the text lengths (in bytes!) to the object size:

```
VBLTestStructure test;

test.mHeader.mBase.mObjectSize = sizeof(VBLTestStructure)
+ sizeof(wchar_t) * test.mExecutingObjectNameLength
+ sizeof(wchar_t) * test.mNameLength
+ sizeof(wchar_t) * test.mTextLength;
```

3.35 VBLDataLostBegin

Description: An error event which indicates the start of a data loss in one of the CANoe queues.

Corresponding object type: BL_OBJ_TYPE_DATA_LOST_BEGIN

Object available starting from CANoe/CANalyzer version 10.0

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mQueueIdentifier	uint32_t	Identifier for the queue where the data lost occurred: 0: Data loss in RT main queue (BL_DL_QI_RT_QUEUE) 1: Data loss in Anlyz main queue (BL_DL_QI_ANLYZ_QUEUE) 2: Data loss in Anlyz and RT main queue (BL_DL_QI_RT_AND_ANLYZ_QUEUE)

3.36 VBLDataLostEnd

Description: An error event which indicates the end of a data loss in one of the CANoe/CANalyzer queues.

Corresponding object type: BL_OBJ_TYPE_DATA_LOST_END

Object available starting from CANoe/CANalyzer version 10.0

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.
mQueueIdentifier	uint32_t	Identifier for the queue where the data lost occurred: 0: Data loss in RT main queue (BL_DL_QI_RT_QUEUE) 1: Data loss in Anlyz main queue (BL_DL_QI_ANLYZ_QUEUE) 2: Data loss in Anlyz and RT main queue (BL_DL_QI_RT_AND_ANLYZ_QUEUE)
mFirstObjectLostTimeStamp	uint32_t	Time stamp of the first lost event in nanoseconds.
mNumberOfLostEvents	uint32_t	Number of the lost events

3.37 VBLWaterMarkEvent

Description: An event which indicates the current water mark level of the CANoe/CANalyzer queue.

Corresponding object type: BL_OBJ_TYPE_WATER_MARK_EVENT

Object available starting from CANoe/CANalyzer version 10.0

Parameter	Type	Description
mHeader	VBLObjectHeader	Common header type. See 3.1.4.

Parameter	Type	Description
mQueueState	uint32_t	Watermark state of the queue: 0: Status normal (BL_WM_QS_STATUS_NORMAL) 1: Status is critical. Data loss may occur soon. (BL_WM_QS_STATUS_EMERGENCY) 2: Status data loss (BL_WM_QS_STATUS_NORMAL)