VLCB Versatille Layout Control Bus

VLCB LongMessage/Streaming Service Specification Service# 17 Stream

Version 0.01

Compatible with CBUS ® 4.0 Rev 8j

0.1 ChangesU

- 2022.12.29 DPH Initial creation.

0.2 To Do

- Make a note of things still needing to be done.
- Would people, while working on a Service doc, note any tests that would be useful in a future Conformance Tool. These are often better thought up at the time of design, rather than a future date. I have added the section below as an example.

0.3 Useful Conformance Tests

- Test short messages with type (BYTE, WORD, U32, JSON)
-

0.1 Table of Contents

0.2 Document History

Date	Changed by	Summary of changes	Service version
2022.12.29	DPH	Initial doc	0.1

1. MERGLCB Stream Service

This document describes the MERGLCB Stream (Long Message) Service. This service is an optional addition to the MERGLCB Minimum Node Specification. Modules wishing to use this Service shall conform to this specification.

2. Introduction / Description / Explanation

MERGLCB messages are generally limited to an opcode and up to 7 data bytes. There is a need to be able to send messages with more data, and in some cases continuous data. This Service is meant to address those needs. Use cases might include:

- Sending data to a Arrivals/Departures display
- Sending a module-description to a Tool
- Sending analog samples to a display

Note however that many transports will have limited bandwidth, or throughput, and the designer of modules and end users must keep those limitations in mind – this service is not meant as a high-speed nor data-intense service. As such, streaming will likely only be useful for short bursts of data sent periodically, or slower continuous streaming. For example, the last example above might be limited to one sample per second, and might be used to display the track-voltage or track-amperage, say. Obviously, one can transmit this information by other means, but this Service might be most convenient once it is set up.

This Service implements a one-way connection between one producer-module and one, or more then one, consumer-module(s).

There is nothing that prevents modules from establishing two-way communication by opening two complementary Streams, one in each direction.

This Service uses an 8-bit unique StreamID to identify the specific stream. The StreamID must be unique across the MERGLCB bus, and therefore has to be allocated by the user. Some StreamIDs may be reserved and dedicated to system-uses, these will be documented in an Appendix, and in the Magic Number Document.

The StreamID used must be agreed to between the modules using the stream. It is used to both identify the stream, and to determine that stream's protocol, if there is one. Again, any protocol must be understood by all of the participating modules. The protocol may be nil, in the case of a simple stream of undifferentiated values, or may be more complicated, for example a JSON document.

This Service uses a Sequence Number to distinguish individual packets making up the Stream. This is to help ensure stream-integrity, and maintain the ordering of packets. This is an 8-bit value. The values 0, and 251-255 are reserved for internal use. The values 1-250 are used for sequencing. For continuous streaming, roll-over of the Sequence Number from 250 to 1 is allowed, and such use is indicated by the Flags.bit0 field in the initialisation packet.

In addition to the above, this Service has optional features, which will be indicated in a Flags parameter field:

- Optional roll-over of the Sequence#, from 250 to 1.
- Optional response messages to Stream/Long-message messages, to enable flow control.

3 Details of Stream Service

3.1 Generic Stream

This Service uses one opcode, DTXC, to both initialise the service, and to send its data.

(DTXC, StreamID, Sequence#, Flags, CRC, Length, data …)

- StreamID - 1-254, must be system unique, and understood by participating modules. It identifies the stream, and determines its protocol. .
- Sequence# - 1-250, must be incremented in each successive data packet. Roll-over from 250 to 1 is permitted if Flags.bit0==1.
- Sequence# 251-255 are used to indicate the last packet of a block, with 1-5 data bytes, respectively.
- Flags - 8-bit options flag
  Bit 0: 0=no sequence# rollover, 1=roll-over from 250 to 1 permitted.
  Bit 1: 0=no responses, 1=mandatory GRSP responses after stream-parts.
  Bit 2-7: reserved.
- CRC - 16-bit cyclic redundancy check value, used to check data integrity, a value of zero means no check is to be made. The CRC is calculated on all the data-bytes of the stream.
- Length - 16-bit length in bytes of the long-message, a value of zero means unlimited data, sequence# roll-over allowed.
- Data - the data to be sent

NB: This is the specification for the message. Developers and libraries may use different implementations. Eg:

- bool longMessage(uint8_t streamID, uint8_t flags, uint16_t length, uint8_t* data)
- void open(uint8_t streamID, uint8_t flags)
  bool write(uint8_t streamID, uin16_t length, uint8_t* data)
  bool read(uint8_t streamID, uin16_t length, uint8_t* data)
  bool close(uint8_t streamID)

  Optional responses, if Flags.bit0==1 these are mandatory:

- **(GRSP, NN, DTXC, 17, Rejected_Busy)**
- **(GRSP, NN, DTXC, 17, InitOk, 0, 0) - no blocksize**
- **(GRSP, NN, DTXC, 17, InitOk, blocksize) - maximum 16-bit blocksize specified**
- **(GRSP, NN, DTXC, 17, Ready)**
-
- **(GRSP, NN, DTXC, 17, CRC_Failure)**
- **(GRSP, NN, DTXC, 17, CRC_OK)**

3.1 Blocking

Many modules will have limited memory for buffers. By replying to the DTXC-initialisation message with **(GRSP, NN, 17, InitOk, blocksize)**, the target node can force the source node to send the data in block-sized pieces, so as to not overflow its buffer(s). If more than one target node consumes this stream, then each may reply with their blocksize, and the source node keep a count of these (n) nodes, and limit its sending to the minimum of those block sizes. After each block is complete, the target-nodes will reply with (GRSP, NN, DTXC, 17, Ready), and the source node must wait for n-responses, or must time out after 1 second, and close the stream.

4 Adaptation to CAN

4.1 CAN Normal Stream Initialisation Message

(DTXC, StreamID, Sequence#==0, Length.hi, Length.lo, CRC.hi, CRC.lo, Flags)

- StreamID - 1-250, must be system unique, and agreed to by participating modules.
- Sequence# == 0 means that this is the initialisation message for this stream.
- Length - 16-bit length in bytes of the long-message, a value of zero means unlimited data, sequence# roll-over allowed. Note, because there are 250 sequence numbers, and 5 bytes per packet, the maximum number of bytes before roll-over of the sequence number is 1259 bytes, and therefore a length of greater than this explicitly allows roll-over.
- CRC - 16-bit cyclic redundancy check, used to check data integrity, a value of zero means no check is to be made.
- Flags - flags indicating optional features.
  Bit 0: 0=no sequence# rollover, 1=roll-over from 250 to 1 permitted.
  Bit 1: 0=no responses, 1=mandatory GRSP responses after stream-parts.
  Bit 2-7: reserved.

Optional responses, if Flags.bit0==1 these are mandatory, and the source node must wait for them:

- **(GRSP, NNhi, NNlo, DTXC, 17, Rejected_Busy)**
- **(GRSP, NNhi, NNlo, DTXC, 17, InitOk, 0, 0) - no blocksize**
- **(GRSP, NNhi, NNlo, DTXC, 17, InitOk, blocksize) - maximum 16-bit blocksize specified**
- **(GRSP, NNhi, NNlo, DTXC, 17, Ready)**
- **(GRSP, NNhi, NNlo, DTXC, 17, CRC_Failure)**
- **(GRSP, NNhi, NNlo, DTXC, 17, CRC_OK)**

4.2 CAN Stream Data Flow Message

(DTXC, StreamID, Seq#, dataN, dataN+1, dataN+2, dataN+3, dataN+4)

- Seq# - 8-bit 1-250, sequential, may roll-over is Flag.bit0==1 in initialisation.
- data - five (5) bytes of data in a packet.
- Seq# 251-255 indicates the last-packet of a block, and has 1-5 data bytes, respectively.

Last-packets of a block:

- **(DTXC, StreamID, Seq#==251, dataN, 0, 0, 0, 0)**
- **(DTXC, StreamID, Seq#==252, dataN-1, dataN, 0, 0, 0)**
- **(DTXC, StreamID, Seq#==253, dataN-2, dataN-1, dataN, 0, 0)**
- **(DTXC, StreamID, Seq#==254, dataN-3, dataN-2, dataN-1, dataN, 0)**
- **(DTXC, StreamID, Seq#==255, dataN-4, dataN-3, dataN-2, dataN-1, dataN)**

Optional responses, if Flags.bit0==1 these are mandatory:

- **(GRSP, NNhi, NNlo, DTXC, 17, Rejected_Busy, 0, 0)**
- **(GRSP, NNhi, NNlo, DTXC, 17, InitOk, 0, 0) - no blocksize**
- **(GRSP, NNhi, NNlo, DTXC, 17, InitOk, blocksizeHi, blocksizeLo)**
  - maximum 16-bit block size specified by consuming-node
- **(GRSP, NNhi, NNlo, DTXC, 17, Ready)**
- **(GRSP, NNhi, NNlo, DTXC, 17, CRC_Failure)**
- **(GRSP, NNhi, NNlo, DTXC, 17, CRC_OK)**

Note, short messages are possible. The total stream may consist of:
(DTXC, StreamID, 0, 0, 1, 0, 0, 0) - length=1
(DTXC, StreamID, 251, x, 0, 0, 0, 0) - single data byte

5 Dependencies on other services

This Service is not dependent on other Services, other than MNS messages listed below. It may use GRSP to Ack/Nak, errors, and in some cases carry stream admin data.

6. Summary of Opcodes

Refer to the MERGLCB Opcode Specification document for details of the opcodes.

Request to Module	**Module's Response**	**Use/meaning**
DTXC	n/a	Streaming opcode, used for command and data
Any	GRSP	OK / Error response with requesting-opcode, and Service-specific data
	CMDERR	Error response
RQSD	SD, ESD	Request Service information

7. Service Specific Modes

No additional operating modes are specified by the Stream service.

8. Service Specific Status Codes

Code	Name	Meaning
255	Rejected_Busy	The stream is already in use, it is not available.
254	Init_Ok	The target node has accepted the initialization message, and is ready for data.
253	Ready	The target node is ready for the next block of data.

252	CRC_Failure	The CRC calculation failed to agree with the valueoffered in the initialization message.
251	CRC_Ok	The CRC calculation agreed with the initialization message value.

Result Code	Service Type	Description	Comment
0xFF	17	Out of range	OUT_OF_RANGE
0xFE	17	Parameter illformed	ILLFORMED
0xFD	17	Protocol sequence error	SEQ_ERROR
0xFC	17	Data check error (checksum, CRC)	DATA_CHECK
0xFB	17	Initiation failure (eg Service, CAN, TCP)	INIT_FAILURE

These Status codes are enumerated from 255 (0xFF) downwards, so that the standard CMDERR codes can be used as necessary.

9. Service Specific Diagnostic Data

- Nbytes_transmitted - the total number of bytes transmitted via this stream.
- nCRC_failures - the total number of CRC mismatches.

10. Service Specific Automatic Power-up Tests

No service specific power-up tests are specified by the Stream service.

11. Service Documentation

Document what StreamIDs are hard-coded, or how to specify the StreamID used for each purpose. A module can use multiple streams, but by mutual agreement with their co-use-modules.

12 Service Data

12.1 Parameters

No parameters are associated with the Stream service.

Address	**Param#**	**Name**	**Usage**	**MERGLCB should set these values**	**Value if service is not supported**
??					

Suggestions: Total-throughput (?packets); #errors

12.2 ESD data bytes

-

13 Discussion / Technical Note

13.1 General use

The essential things to realise is that:

- Streams are unidirectional, so use two streams for bidirectional communications.
- The cooperating nodes must agree on a protocol.
- A Stream can be consumed by more than one target node. If these target nodes respond with different block sizes, then the source-node must use the minimum.
- A Stream can be closed and opened again to use multiple protocols, or protocol-parts. Identification of these are up to the nodes to agree to.
- Typically a steam would include a protocol-identifying byte at the beginning of the stream.

Examples of possible protocols:

- Sending bytes, words and uint32_t:
  (DTXC, StreamID, 0, 0, 2, 0, 0, 0) - length=2, type and value
  (DTXC, StreamID, 252, BYTE, x, 0, 0, 0) - single byte
  (DTXC, StreamID, 0, 0, 3, 0, 0, 0) - length=3, type and value
  (DTXC, StreamID, 253, WORD, x.hi, x.lo, 0, 0) - single word
  (DTXC, StreamID, 0, 0, 5, 0, 0, 0) - length=5, type and value
  (DTXC, StreamID, 255, UINT32, x.3, x.2, x.1, x.0) - single uint32
- Sending JSON:
  (DTXC, StreamID, 0, 0, 25, 0, 0, 0) - length=25
  (DTXC, StreamID, 1, JSON, ‘{‘, ’n’ ,’a’, ‘m’) - “*JSON*{nam”
  (DTXC, StreamID, 2, ‘e‘, ’:’ ,’M’, ‘E’, ‘R’) - “e:MER”
  (DTXC, StreamID, 3, ‘G‘, ’,’ ,’u’, s, ‘e’) - “G,use”
  (DTXC, StreamID, 4, ‘:‘, ’R’ ,’u’, ‘l’, ‘e’) - “:Rule”
  (DTXC, StreamID, 255, ‘s‘, ’ ’ ,’O’, ‘k’, ‘}’) - “s Ok}”

13.2 Use in bootloading

Bootloading and Loading is a complex subject, and is very dependent on the processor and compilers used. Many systems have standardised on:

- .hex -hexfiles
- .bin - binary files
- .elf - executable and linkable format files
- .eep - eeprom files

to carry the results of a compilation, and linkage, to the memory of the processor. In addition to writing to the flash, writing to the eeprom or other non-flash storage, such as processor configuration bits has to be accommodated. This makes having a generic protocol difficult, but likely not impossible.

The steps are:

1. Generate the appropriate file(s) for uploading to the target processor.
2. Identify them to a Loader Tool.
3. Identify the target processor.
4. Send the file(s) to the processor via the DXTC message.

Of necessity, the Tool and the processor will need to agree on a protocol. One such one is suggested:

1. The Loader sends a BOOT message to the target node, and the target node enters its bootloader.
2. The Loader sends the file by sending an initialisation DXTC message and data, with the first line of the file:
   (DXTC, SN, 0, length, 0)
   (DXTC, SN, 1, HEADER, …)
   (DXTC, SN, EOD, …)
   The target node replies with a InitOK and a maximum blocksize:
   (GRSP, NNhi, NNlo, DTXC, 17, InitOK blocksize)
3. The Loader sends the rest of the file as a series of blocks, of maximum blocksize:
   (DXTC, SN, 0, length, 0)
   (DXTC, SN, 1, DATA, …)
   (DXTC, SN, EOD, …)
   The target node indicates readiness for the next block by sending a GRSP(Ready) message:
   (GRSP, NNhi, NNlo, DXTC, 17, Ready)
4. At the end of the file, the Loader sends a final block with CRC:
   (DXTC, SN, 0, length, 0)
   (DXTC, SN, EOD, CRC, crc)
   The target responds with CRCOk or CRCFailure:
   (GRSP, NNhi, NNlo, DXTC, 17, CRC_Ok) or
   (GRSP, NNhi, NNlo, DXTC, 17, CRC_Failure).