OpenLCB Standard	
Stream Transport	
Jan 30, 2021	Preliminary

1 Introduction (Informative)

This specification defines the protocol for transmitting unidirectional streams of data between two nodes on an OpenLCB bus.

2 Intended Use (Informative)

Streaming can be used to move large amounts of data (kilobytes and up) across a OpenLCB implementation. Streaming transport can also be used when the data transfer is unknown or indefinite in its volume or persistence.

References and Context (Normative)

This specification is in the context of the following OpenLCB Standards:

- The OpenLCB Message Network Standard, which defines the basic messages and how they interact. Higher-level protocols are based on this message network, but are defined elsewhere. The Message Network Standard defines the encoding of stream data message on CAN physical layer, which is referenced here.
- The OpenLCB Memory Access Configuration Protocol Standard, which will be used as an example of an application-level protocol that interacts with the Streaming protocol for data transport.

3 Definitions (Normative)

Stream is a uni-directional sequence of bytes transmitted from one OpenLCB node to another OpenLCB node. There is no out-of-band information that can be sent along with the payload using the Streaming protocol. There is no restriction on the total number of bytes sent, or the time that the stream can be open for sending data.

Source node is the OpenLCB node which is sending the data (originating the stream).

Destination node is the OpenLCB node which is receiving the data.

Source Stream ID (**SID**) is a one-byte (8-bit) identifier (range [0..254], reserved [255]), assigned by the source node at its discretion to identify the stream.

Destination Stream ID (**DID**) is a one-byte (8-bit) identifier (range [0..254], reserved [255]), assigned by the destination node at its discretion to identify the stream.

Stream Content UID is a non-zero 6-byte globally unique identifier assigned to specify the type (format and purpose) of the data being transmitted. Stream Content UIDs are assigned in a similar fashion as Node IDs, but from a different numeric space.

(Max) Buffer size is an unsigned 2-byte value (range [1..65535]) specifying the maximum number of bytes in-flight; that is, the maximum number of bytes the source node may send without receiving a reception acknowledgement from the destination node.

4 Message Formats

In the following, the “common MTI” column specifies the MTI value to be used when communicating in OpenLCB common format. The Common MTI is an abstract numeric description intended as a normative guide to the construction of concrete message formats over specific physical transport media.

4.1 Stream Initiate Request

Stream Initiate Request is an addressed message sent by the Source node to the destination node to request creating the stream. This is the only method to create a stream. The Destination Stream ID is only to be suggested if previously negotiated with the destination node using some other prerequisite protocol.

Data content:

Name	Dest ID	Event ID	Common MTI	Data Content	
Stream Initiate Request	Y	N	0x0CC8	6 or 12 bytes	
				Byte 0	Suggested Destination Stream ID, else 0xFF if none is to be suggested.
				Byte 1	Source Stream ID, value 0xFF is reserved.
				Bytes 2-3	Max Buffer Size (proposed by source).
				Byte 4	Flags: LSB (bit 0): Deprecated, send as 0, check upon receipt. bit 1-7: Reserved, send as 0, check on receipt.
				Byte 5	Additional Flags: Reserved, send as 0, ignore on receipt.
				Bytes 6-11	Optional: Stream Content Type.

4.2 Stream Initiate Reply

Name	Dest ID	Event ID	Common MTI	Data Content	
Stream Initiate Reply	Y	N	0x0868	6 bytes + optional error string	
				Byte 0	Destination Stream ID, value 0xFF is reserved.
				Byte 1	Source Stream ID, value 0xFF is reserved.
				Byte 2-3	Max Buffer Size (final value), or zero if the request is rejected.
				Bytes 4-5	Error code: 0x8000 for accept, no errors 0x1000 Permanent error, specifically: • 0x1040 streams not supported • 0x1020 source not permitted • 0x1010 unimplemented • others reserved, do not send 0x2000 temporary error, specifically: • 0x2040 unexpected, out of order, or inconsistency in the message or frame sequence received. • 0x2020 buffer unavailable • others reserved, do not send 0x0001 may be logically or'd with any of the above error codes to indicate the incident is available in a log (see the Logging protocol for

					more information).

4.3 Stream Data Send

Name	**Dest ID**	**Event ID**	**Common MTI**	**Data Content**	
Stream Data Send	Y	N	0x1F88	1 – <Max Buffer Size + 1> Bytes	
				Byte 0	Destination Stream ID, value 0xFF is reserved.
				Byte 1 ...	Stream payload bytes.

4.4 Stream Data Proceed

Name	**Dest ID**	**Event ID**	**Common MTI**	**Data Content**	
Stream Data Proceed	Y	N	0x0888	2 Bytes	
				Byte 0	Destination Stream ID, value 0xFF is reserved.
				Byte 1	Source Stream ID, value 0xFF is reserved.

4.5

4.6 Stream Data Control

Name	Dest ID	Event ID	Common MTI	Data Content	
Stream Data Control	Y	N	0x08A8	2 or 6 Bytes	
				Byte 0	Destination Stream ID, value 0xFF is reserved.
				Byte 1	Source Stream ID, value 0xFF is reserved.
				Byte 2	Type (bits 0..3): 0x0: reserved 0x1: ping 0x2: pong 0x3: completed 0x4: abort Flags (bits 4..7): 0x1: 1 if this message is from the stream source to stream destination, 0 if this message is from the stream destination to the stream source. 0x2: 1 if payload count is valid, 0 if payload count is not valid. 0x4: reserved 0x8: 1 if stream closed, currently only valid for a pong type, all other types must send 0 and ignore upon receipt.
				Bytes 3-5	Number of payload bytes transferred in total (sum of all data segments) % 2^{24}.

5 States (Normative)

A stream can be in one of the following states:

Non-existent or **closed**. Between the nodes <Source Node ID> → <Destination Node ID> both the Source Stream ID and the Destination Stream ID have never been used or the Stream according to the last use was completed or terminated.

Announced. The Source node has sent the Source Stream ID to the Destination node via some higher-level protocol, such as in a Memory Configuration Protocol datagram 'Read Stream Reply', or 'Write Stream Command'.

Initiated. The Source node has sent a Stream Initiate Request to the Destination node with the Source Stream ID, and is waiting for a response.

Open. The Destination node has sent a Stream Initiate Reply to the Source node with the Source Stream ID and the Destination Stream ID and the error code set to Accepted.

6 Interactions (Normative)

6.1 Opening a stream with announcement

In this method the streaming protocol works together with a higher-level protocol. It is the higher-level protocol that requests the transfer, and defines the format, contents and possibly the length of the stream payload. As part of handling the higher-level protocol, the Source node has to know the Destination node ID, and allocate a Source Stream ID for that Destination node that is in the Closed / Non-existent state. The Source node transmits the Source Stream ID to the Destination node in an addressed message of the higher-level protocol. The Source node then waits for a reception acknowledgement of the higher-level protocol (such as a Datagram Received OK) from the Destination node. If this acknowledgement is positive, the Stream transitions to the Announced state, if it returns an error, the Stream is considered to be in the Closed state. If the Destination node can not handle the streaming protocol, it should return an error in the higher-level protocol.

After the acknowledgement of the reception of the announcement message, the Source node sends a Stream Initiate Request to the Destination node, with the same Source Stream ID as in the announcement message. The Source node may pass or omit the Stream Content Type in the Stream Initiate Request, as the higher-level protocol specification requires. The Source node proposes a Max Buffer Size for the transmission. Sending this message transitions the stream to the Initiated state.

The Destination node is then required to send a Stream Initiate Reply, or one of the general error messages defined in the Message Network Standard, such as Optional Interaction rejected, or Terminate Due To Error. When possible, a Stream Initiate Reply should be sent instead of a general error message.

To accept the stream and transition to Open state, the Destination node must allocate a Destination Stream ID that's not currently in use with the same Source node, reply with a Stream Initiate Reply message, with the error code having the Accept bit set, the Source Stream ID set to the stream ID sent by the Source node, the Destination Stream ID filled in, and the Max Buffer Size set to a non-zero value that is at less, than or equal to what was proposed in the Stream Initiate Request. Sending this message transitions the stream to Open state, with the Max Buffer Size as entered in the Stream Initiate Reply message.

To reject the stream, the Destination node must send a general error message, or a Stream Initiate Reply message with the appropriate Source Stream ID, zero Max Buffer Size and the error code set to a value with the Accept bit clear and one of the Temporary error or Permanent error bits set. When possible, a Stream Initiate Reply should be sent instead of a general error message. When sending a general error message, when possible, the MTI value should be set to the MTI of Stream Initiate Request. A rejected stream is transitioned to the Closed state. If needed, the interaction may be re-started using the higher-level protocol.

6.2 Opening a stream without announcement

This method may only be used if the interaction is started by the Source node.

The interaction begins with the Source node allocating a new Source Stream ID, and sending a Stream Initiate Request to the Destination node. This message must carry a 6-byte Stream Content UID, which specifies for the destination node the content and format of the data carried, and transitions the stream to the Initiated state. From this state onwards the interaction continues as in described in Section 6.1.

If the Stream Content UID is unknown to the Destination node, it should use the Permanent Error, Unimplemented error code in the rejection message.

6.3 Data transfer

Once a stream transitioned to Open state, the Source node may start sending payload bytes using Stream Data Send messages. The source node may send up to Max Buffer Size bytes using one or more Stream Data Send messages, without needing to wait for acknowledgement from the destination node.

When the Destination node is ready to receive another Max Buffer Size bytes, it sends a Stream Data Proceed message to the Source node. This signals to the Source node that it may send another Max Buffer Size bytes. Each Stream Data Proceed message, irrespectively of when it was sent, extends the window of allowed payload bytes to be sent by Max Buffer Size.

If the Source node has no more data to send, but wants to keep the stream open, it may, but is not required to periodically send Stream Data Send messages with zero payload bytes to ensure that the destination node and any gateways in the transfer path are still aware of the stream transmission.

6.4 Closing the stream

The Source node signals completion of the transfer by sending a Stream Data Complete message to the Destination node, providing the Source Stream ID and the Destination Stream ID with it. Optionally the Source node may add a 4-byte value describing the total number of bytes sent in the stream. If the Source node does not want to specify this value, it may omit the bytes from the Stream Data Complete message or may send 0xFFFFFFFF to specify unknown length.

The Stream Data Complete message transitions the stream to Closed state.

The Stream is closed if the Destination node sends a Terminate Due To Error or Optional Interaction Rejected message with no MTI value or with the MTI value of Stream Data Send; or if the Source node sends a Terminate Due To Error or Optional Interaction Rejected message with no MTI value or with the MTI value of Stream Data Proceed. When possible, the specific MTI value should be used.

6.5 Special provisions for Gateways

A Gateway needs to be prepared for buffering up to Max Buffer Size bytes for each open stream. It may reduce the Max Buffer Size in the Stream Initiate Request message if the buffer size available is smaller than proposed. A Gateway must not modify the Max Buffer Size value of a Stream Initiate Response message.

A Gateway may repackage the stream payload into fewer or more Stream Data Send messages than received, as the underlying network requires. A Gateway must emit a Stream Data Send message with empty payload after its buffer is drained, if it has received such a message.

A Gateway may delay forwarding Stream Data Proceed messages until its internal buffer is drained, but it must not drop any of them.

A Gateway must delay forwarding the Stream Data Complete message until its internal buffer is drained.

A Gateway must drop all data in its internal buffer and all delayed messages if the Stream is terminated with an error message of Terminate Due To Error or Optional Interaction Rejected.

7 Adaptation to CAN Transport (Normative)

This section describes the CAN implementation of stream transport message formats.

7.1 Stream Data Send

Stream Data Send messages are sent on a CAN network using the Frame Type of 7 (Stream Data), the Destination node alias and the Source node alias in the CAN header, and the Destination Stream ID in the first data byte. The remaining data bytes can e filled with stream payload (up to 7 bytes per message):

Name	Dest ID	Event ID	Header Format	Data-part Content
Stream Data Send	Y	N	0x1Fdd,dsss	Byte 0: Destination Stream ID Byte 1...: payload bytes

7.2 All Other Messages

All other messages are sent as described in the OpenLCB Message Network Standard. Note that the Stream Initiate Request message may need to be sent in two CAN frames if it contains a Stream Content UID. The format for multiple frame CAN messages is described in the OpenLCB Message Network Standard.

that support the Streaming protocol must keep track of the Stream states

Optionally, the destination can request that the source start a transfer. This uses some mechanism not discussed here, e.g. Datagram messages.

The source sends an "Stream Initiate Request (Addressed)" to the destination. It carries a "Max Buffer Size" value (2 bytes), a “Type Included” boolean flag (1 bit in a byte; see below for meaning) and a "Source Stream ID" (1 byte) in the data section. The combination of source, destination, and Source Stream ID must uniquely identify this stream transmission.

If the destination node does not wish to receive the stream, it returns a "Stream Initiate Reply (Addressed)" marked “reject”, with a "Max Buffer Size" value (2 bytes) of zero, the “Type Included” boolean flag (1 bit in a flag byte), "Source Stream ID" (1 byte) and "Destination Stream ID" (1 byte). The message includes flags set to represent exactly one of several conditions:

“Permanent error” - This node does not process streams, will not accept a stream from this source, or for some other reason will not ever be able to accept this stream. The Stream Initiate Request should not be retransmitted. Optionally, the node can mark the reply with one or more of several conditions:

“Information Logged” - the node supports the logging protocol and information was logged for later retrieval.

“Invalid Stream Request” - something made the stream request improper, such as longer than the max permitted length. Proper requests might be acceptable.

“Source not permitted” - streams from this source will never be accepted

“Streams not accepted” - this node will not accept stream requests under any circumstance. A node can also reject the interaction instead of sending Stream Initiate Reply with this code.

“Buffer shortage, resend” - The node isn't able to receive the stream because of a shortage of buffers. The sending node should resend at its convenience.

“Stream rejected, out of order” - Should not happen, but an internal inconsistency was found in the CAN frames making up a stream. Sender can try again if desired.

If the destination node wishes to receive the stream, it returns a "Stream Initiate Reply (Addressed)" marked “accept”, with a "Max Buffer Size" value (2 bytes), the “Type Included” boolean flag (1 bit in a byte), "Source Stream ID" (1 byte) and "Destination Stream ID" (1 byte). The "Max Buffer Size" is less than or equal to the value in the Initiate Stream Request, and is the negotiated buffer size for this transfer. If it's zero, the request to start the stream has been rejected, and the exchange is over. The source can try again later.The Source Stream ID is the same as the value in the Initiate Stream Request, and is returned for identification and the convenience of the source. The destination doesn't do anything with it except return it. The source can use it to match up multiple operations, as a way of identifying buffers, or for any other purpose.The Destination Stream ID is used to tag the data sent to the destination. It has no meaning to the source. The destination can, but need not, use it to associate the stream data with a particular buffer or usage. Multiple simultaneous streams can use the same Destination Stream ID value.

The source starts sending bytes using Stream Data Send (Addressed) messages, each carrying up to the message size limit on the particular wire protocol and the Source and Destination Stream ID. After sending exactly Max Buffer Size bytes in one or more messages, the source pauses.

If the “Type Included” flag was true during initialization, the first 6 bytes of the data are a unique data-type indicator. These are allocated the same way that Node IDs, etc, are allocated, but from a separate name space. If that “Type Included” flag is false, some higher-level protocol must identify the data-type of the stream data. The idea is that a stream is a lot of data; there's not much use for one otherwise because of the setup overhead (code and time). So six bytes for a stream type identifier isn't a large cost (unlike e.g. a datagram, where it would be a 10% overhead). A UID as a Stream type ID has the advantage that it's not ever going to collide, so people developing protocols don't have to coordinate it (e.g. useful for future expansion, where a protocol can be locally developed and then deployed more widely). And it still fits in the 1st CAN Frame of the stream, which simplifies what nodes have to do to figure out what type of (unsolicited) data this is. On the other hand, streams have a stream ID, so that a protocol can use some other mechanism (messages, datagrams, or even other streams) to pass the information about what a specific new stream means. That means the type info at the start of the stream isn't as necessary as in e.g. datagrams. (We really wanted to avoid situations like "The next datagram you receive carries the data for this request", because "next" is a hard concept to ensure in code that's doing several things independently)

On CAN, the Stream Data Send message does not carry the Destination Stream ID field. Messages to CAN get split into frames and forwarded without the field. Messages from CAN have to have it inserted by the gateway as part of the translation process. The gateway can do this as part of the (optional) buffer accumulation from frames into larger transfers.

Upon receiving Max Buffer Size bytes via one or more messages, the destination sends a Stream Data Proceed (Addressed) message to the source, carrying the Source Stream ID and Destination Stream ID. This tells the source that there is enough buffer space available that another Max Buffer Size bytes can be sent. The destination can also set flag bits (error codes) in this message to indicate that no more data should be sent and the transmission should be terminated early.

The destination may, but is not required to, send a Stream Data Proceed (Addressed) message to the source before receiving the full count of Max Buffer Size bytes. In that case, the source does not stop transmission after sending Max Buffer Size bytes, but continues for an additional Max Buffer Size bytes of transmission.

When the last data has been sent, the source sends a Stream Data Complete message carrying the Source Stream ID and Destination Stream ID to indicate that all data has been sent. Optionally, this can carry a four-byte length of the stream data transferred for checking. A zero value, if present, means the length is unknown. When this message is received by the destination, the transfer is complete.

8 Background Information

8.1 Stream Content Type Identification

Stream senders can, but don't have to, identify the content type (format, meaning, etc) of the data stream at the front. As part of their private protocol, they can use whatever structure they'd like for the data.

General content types can be defined using unique identifiers. The “Type Included” flag identifies that the first eight bytes of stream data are to be interpreted as stream content type information. These can be allocated via the same mechanisms as EventIDs. Well-known ones will be published by OpenLCB. These identifiers are recorded in a separate worksheet.

8.2 Buffer Size

The buffer size can be up to 216-1 bytes = 64KB-1. This is driven by the size of the initiate messages, which we want to keep in a single CAN frame. Although some transport mechanisms might be able to profitably use larger buffers, it seems unlikely that a layout control bus will need to have a higher message efficiency or lower latency than this size will allow. Should that turn out to be necessary in the future, an alternate form for the stream initialization messages can be defined for those large-message nodes. (It's unlikely they're running over CAN)

8.3 CAN Discussion

On a CAN segment, the data limit is 8 bytes. This requires use of as little control information as possible, so that as many of those eight bytes can be used for data as possible.

OpenLCB/S9.6 CAN frames can hold the source and destination aliases in the header. A dedicated “stream data transfer” format can also be coded entirely in the header. This then allows 8 bytes of data per transfer if the Source Stream ID and Destination Stream ID need not be carried. By specifying that only one stream can be transferred on a CAN link between any two nodes at a time, no Stream ID would be needed. Although initially consider, this it thought to be too inflexible. Instead, we chose a format where the Source Stream ID is carried, reducing the payload to seven bytes per frame. This requires that the Source Stream ID be unique among source-destination node pairs, without relying on the Destination Stream ID to be part of the unique stream identification. Since a node knows that it's originating the stream, this can be ensured in advance for point-to-point connections. Gateways will need to track the Destination Stream ID to restore it to the full format. Since they are tracking the stream in any case for buffering purposes, this is thought to be not a large problem.

8.4 Implementation Notes

The "Stream Data Proceed" from the destination is clearance to send another buffer-size-worth of data. To achieve better performance, the destination can send it before receiving the entire buffer-size-worth of data, as soon as it has room to receive what's already been OK'd plus one more buffer size. For example, a destination with a 4kB buffer could reply with Max Buffer Size of 2K, followed by an immediate Stream Data Proceed, to do single overlap of the transfer.

Intermediate nodes need to be able to handle transfers, and therefore need permission to lower the Max Buffer Size on the outbound Stream Initiate Request message. The length in the returned Stream Initiate Reply can't be changed, as the destination need to know when to send clearance for another buffer's worth of data.

A CAN ↔ Ethernet bridge might receive several kilobytes of data from the Ethernet side at a time. It's then responsible for breaking that up into CAN frames and forwarding it. It can reduce the buffer size of transfers if need be to ensure there's a place to store the data while this is done.

Table of Contents