VLCB Versatile Layout Control Bus

VLCB Event Model

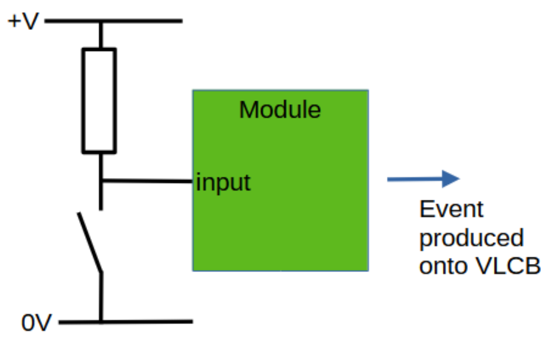
0.1 Table of Contents

0.2 Document History

Date	Changed by	Summary of changes
2024.02.25	DPH	First draft copied from original Event Teach Service specification

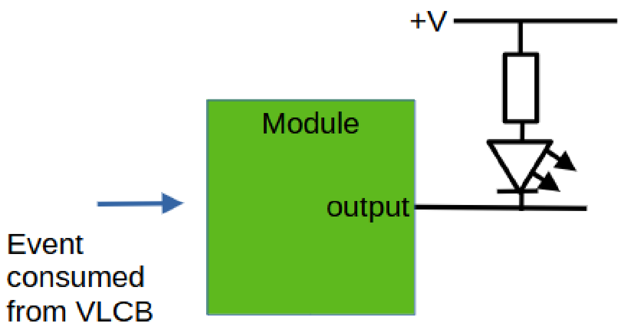
1 Events Introduction

Event messages are a major method of sending information from one node to others. They are commonly used in the Producer-Consumer Network Model. The information is transmitted onto the bus by sending a message containing an ***event identifier*** representing a ***specific change of state*** on the layout. The event conveys the state of the Producer as being either ON or OFF. Most often this is a change in state of a layout accessory, such as an occupancy detector, but is not limited to this. One can think of this event identifier as analogous to a number tag on a wire between two nodes. The nodes must agree on the meaning of the event identifier, for example it might represent the change of state of a switch, whether it is open or closed, or the state of occupancy of a piece of track. This change is denoted by a message containing the event identifier which is sent by the ***Producer Node***.



In addition, a ***Consumer Node*** may use the receipt of the event message to perform some action, most commonly on an attached accessory. For example the action might be turning a LED on or off, or changing the position of a point (turnout).

For events to be useful, two or more nodes have to agree on the meaning of a particular event identifier. This is done by 'teaching' the nodes to use the same event identifier so that the producer can send it on a specific change of state, and the consumers can use it to perform some resulting action.



One nice feature of events is that when the producer sends its event message it does not need to know whether one, many, or no consumer nodes are receiving or consuming its message. It means that the producer and consumer nodes are independent, and that additional consumer nodes can be added at a later date without having to change the producer node, or any of the other consumer nodes.

As shown above, events can be considered to be:

- **Produced** by a node due to some action, such as a button press, track occupancy, a specific time;
- **Consumed** by a node and causes a action, a lamp lighting, a bell ringing, a point moving;
- **Consumed and Produced,** some events can be both produced and consumed, e.g. an event representing a specific time might be produced by a clock module and consumed by the same node to set the current time to that specific time. This is also referred to as "consume own events".

How a module uses events will most often be determined by the node's developer. However, in some cases the user may be able to change an event's capabilities by writing to its associated event variables. This must be documented in the manual accompanying the module.

Other Event-related messages include:
- Messages to put a node into, and out of, **Learning mode**, and to teach events to that node;
- **Request messages** for the status of an event from its producer;
- **Administrative messages** to clear, count, and determine event-space available in a node.

1.1 Long and Short Events

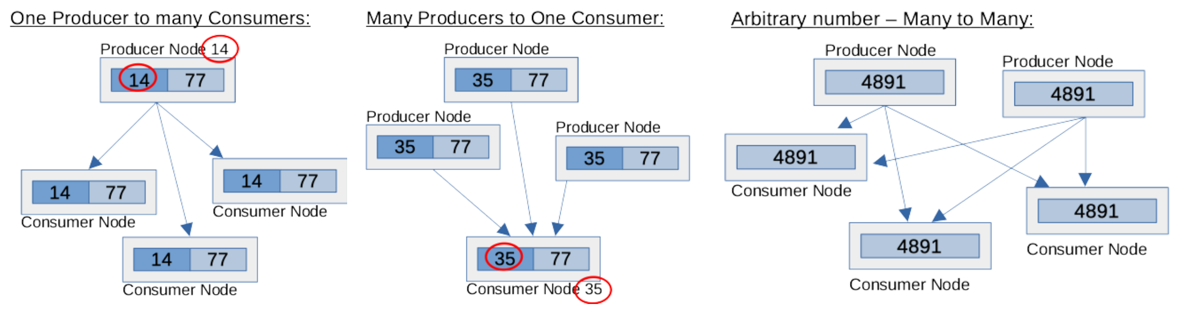
There are two separate Event-message forms:
- Long events, with a 32 bit (4 byte) event identifier consisting of an event number and event node number
- Short events, with a 16 bit (2 byte) event identifier consisting of an event number.

These forms are carried by two different sets of opcodes, and hence have two non-overlapping spaces.

The specific event numbers (EN) can be considered to be specific arbitrary numbers, but can be interpreted as the user wants to embed information into the events. A user may choose to use specific ranges for specific purposes. For example all points might be labelled 100-199, and events assigned to match.

Classically short events have been used in this way, but interprets long events as having two fields, NN and EN, which represent the producer node's NN and an event number. This is convenient as a producer node can construct its long events by combining its NN with a set of ENs, which are typically assigned to its inputs. This can be used well with one producer to one consumer, or one producer to many consumers relationships. One advantage of this interpretation is that the NN field can be used to indicate the producer node.

However, the above is only one interpretation, and others can be considered. For example, the NN in the NN:EN might be interpreted as the consumer node NN. This interpretation allows many producers to one consumer, with the NN identifying the target or consumer node. Conversely, one can simply assign an arbitrary number to the whole NN:EN and create a many producer to many consumer relationship. How events are used is up to the modeller. See below.



Of note, using arbitrary event numbers lets one divide them up as one wishes, so for example:

- Make all buttons range from 1000-1999, all points 2000-2999, all signals 3000-3999;
- Number buttons, points and signals based on track mileage: a point at Mile 100 might be 10012, a signal 10022, and its controlling button 10052.

Whatever scheme is used, it is very important to keep excellent records.

1.1.1 Long Events

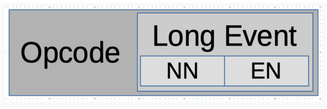
Long Events identifiers are 32-bits (4 bytes) long. Any long-event can be sent by any node.

The long event identifiers are usually interpreted, and used, as having two 16 bit (2 byte) parts:

- ***eNN - the event's-Node-Number***, two-bytes or 1-65536 (0x0001-0xFFFF)
  NB: an eNN of 0 must not be used.

- ***EN - the event-number (EN)***, two-bytes or 0-65536 (0x0000-0xFFFF)

The combination is often written as NN:EN.



And since a NN cannot be 0, the range of NN:EN is 0x00010000 to 0xFFFFFFFF, or 65536-4294967295(!)

Although Long events can be used in many-to-many relationships, VLCB recommends that they should only be used for one producer node to one, or many, consumer node relationships. By having the event identifier contain the producer-NodeNumber, tracing the origin of the event is made easy. While more than two producer nodes can send the same Long event message, this defeats the tracing capability, and is often referred to as spoofing. Event spoofing has been discouraged but may be useful for some modules and for some situations.

However, users are free to interpret the 32 bits of long-message in any way they wish. For example, one might use the NN field as the consumer's NN, and therefore allow a many producers to one consumer relationship. It is obviously important to record these choices for the user's future reference.

1.1.2 Long Event Messages

- ACON - Accessory On
  This message represents the change of state to 'On'

  Data field: eNN:EN

- ACOF - Accessory Off

  This message represents the change of state to 'Off'

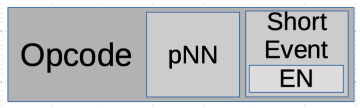
  Data field: eNN:EN

1.1.3 Short Events

Short event identifiers are 16-bits (two bytes), and are represented by a single part:

- **Event Number (EN)**, 2-bytes or 0-65535

They have a distinct set of opcodes (ASON/ASOF).



Note that the NN in the message is usually the producer NodeNumber (pNN), and is included in the messages as a separate field. This is useful for tracing the origin of a specific message, as it can differ for each producer node sending the same Short event message. NOTE, only the EN field shall be used to match the events stored in the node, and Consumer nodes shall ignore the pNN when processing the event.

Note to implementers, short messages can be recorded as 00:EN, where the top two bytes are zero, and since long events (NN:EN) cannot have a NN of zero there will not be a conflict. This allows both long and short events to be stored in one table, and share the same search function.

1.1.4 Short-Event Messages

- ASON - Accessory Short-On
  - This message represents the change of state to 'On'
  - Data-fields: producer NodeNumber, EventNumber
- ASOF - Accessory Short-Off
  - This message represents the change of state to 'Off'
  - Data-fields: producer NodeNumber, EventNumber

These message forms include the producerNodeNumber (NN) as a separate field.

1.2 Bus Data

Only the Opcode and NN:EN are transferred between modules on the bus. The configuration at the Producer to specify what happened to produce the event and the configuration at the Consumer what actions to perform when the event is received is specified using Event Variables (EVs) at the Producer and the Consumer. See 3 Event Variables (EV).

Note: The Event Variables are **not** transferred on the bus as part of the event messages.

1.3 Event ON and OFF

That the concept of 'ON' and 'OFF' is up to the user and may vary with use. For example, 'ON' may represent a button in the down position, while 'OFF' represents the button in the up position, or vice versa. A module may have let this be set with the Event Variables.

1.4 Events With Data

The reader will have noted that there is not just ACON, ACOF, ASON, and ASOF messages, but also similar ones: ACONn, ACOFn, ASONn, and ASOFn. These allow events to be sent with associated data, and could be used, for example, to send a potentiometer position, which might be used to determine how fast an LED flashes.

Opcode	**Number of data bytes**
ACON1, ACOF1, ASON1, ASOF1	1
ACON2, ACOF2, ASON2, ASOF2	2
ACON3, ACOF3, ASON3, ASOF3	3

1.4.1 Distinguishing Events With and Without Data

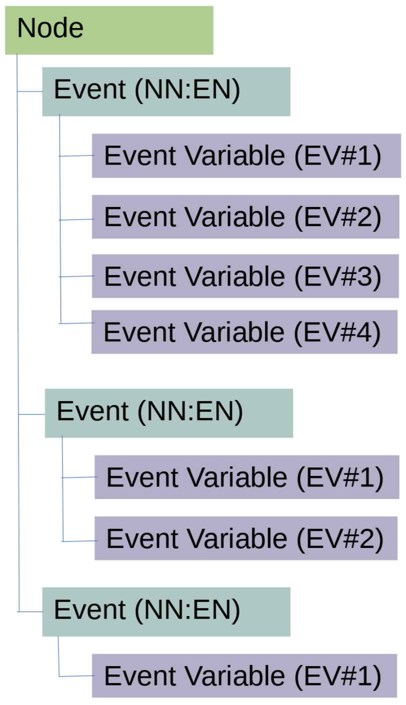
How a module responds to these is up to the designer of the module, ie it is application dependent, but the following are recommended:

a) If a module has been taught an event which requires associated data, and that module receives the event without data then it should send a GRSP error. The GRSP message includes the opcode of the offending messages and an EN_DATA error-code.

b) If a module handles events with no data, it can optionally handle events with data in the same way, but this must be DOCUMENTED in the module's manual.

2 Event Variables (EV)

Event Variables (EVs) are a set of 1-byte values that allow a user to configure a module's Event handling behaviour. Different Events may have differing numbers of associated Event Variables, and this will depend on module design.



Representation of a node with Events and Event-Variables

When a Consumer event is received by a module, the module uses the EVs associated with that event to determine what actions are to be taken by the module. For example, an event variable might determine which digital output to change, the polarity of the output, or the flash rate of a signal.

The EVs associated with an Event may also be used to determine under what conditions a module may produce/transmit that Event. VLCB does not impose any constraint on the usage of EVs however a software library implementing VLCB may impose restrictions on the format and/or use of EVs.

3 Event Teaching

As stated in the introduction, for events to be useful, two or more nodes have to agree on the meaning of a particular event identifier. This is done by ‘teaching’ the nodes to use the same event identifier so that the producer can send it on a specific change of state, and the consumers can use it to perform some resulting action.

Events are taught to modules by configuring EVs for a specific EN/NN event identifier.

3.1 Teaching Long and Short events

Note that there are no separate messages to teach Long and Short events. However, since Long events (pNN:EN) must not have a pNN of zero, Short events can be taught by sending the event identifier as (00:EN). How this is handled in a module is implementation specific, and not part of this document. However, three common methods would be to have a single table and maintain the (00:EN) form for Short events; tag the event identifiers; or maintain two tables, one each for Long and Short events.

Since the only way to teach events is via EVLRN or EVLRNI, which include an event, an event-variable index, and an event-variable value, when teaching an event one also teaches an event-variable at the same time.

3.2 Module event capacity

Information about the quantity of events stored by a module can be found:

- Parameter 4 returns the maximum possible number of events that can be stored by a module.
- Parameter 5 the maximum number of event variables per event
- The EVLNF response from a module when sent an NNEVN request indicates the number of event slots remaining in the module.
- The NUMEV response from a RQEVN request is the number of events currently stored by the module.

Note that the value in EVLNF (remaining space) added to the value in NUMEV (used space) must be less than or equal to Parameter 4 (maximum number of spaces).

3.3 Reading Events

A configuration tool may read all the events from a module using the NERD message. The module responds with an ENRSP message for each event stored within the module.

The module shall send the ENRSP messages at a rate allowing the configuration tool to receive and handle these messages. The module shall transmit these messages with at least 10 ms between each message.

3.4 Reading Event Variables

A configuration tool may read all the event variables associated with an event from a module using a sequence of REQEV messages.

A request for EV index 0 returns all EVs via multiple EVANS messages.

An attempt to read an EV with an index beyond the number of EVs for the event will return a CMDERR(6), and a GRSP(6,REQEV,ev#,0).

3.5 Writing Event Variables

In order to write an EV the module must first be put into Learn mode using MODE(NN, Learn) or NNLRN message.

A configuration tool may then write an EV with the EVLRN message. The configuration tool must specify the event NN/EN, event variable index and event variable value.

The meaning of the event variable values is module dependent and must be documented by the module designer.

If an invalid event variable index is specified the module will return a CMDERR(6), and a GRSP(NN,6,EVLRN,0,0).

If an invalid event variable value is specified the module will return a CMDERR(11), and a GRSP(NN,11,EVLRN,0,0).

After a successful write of EV the module will respond with WRACK, and a GRSP(NN,OK,EVLRN,ev#,actual-value-written).

After writing the events and event variables the module should be reverted to Normal mode using the MODE(NN, Normal) or NNULN message.

3.6 Removing an Event

A configuration tool may remove an event from a module using the EVULN message. The module must be put into Learn mode using the NNLRN message prior to removing the event. The event to be removed is specified with the Node Number and Event Number within the EVULN message.

Node responds with GRSP(NN,OK,EVULN,0,0), or GRSP(NN,NAK,EVULN,0,0) if not found.

All event variables for the event are also removed.

After removing the events the module should be reverted to Normal mode using the NNULN message.

3.7 Removing an Event Variable

There is currently no dedicated way to remove a single EV. If a module is required to support EV removal as opposed to complete event removal then it should dedicate an EV value to indicate that the EV is not used.

It is recommended that if all EVs for the event are set to be unused then the event is also removed.

4 Data messages

Note that ACDAT, RQDAT and ARDAT are not events and are not included within the Event services.

There is no specific change in the teaching process to teach events with data. Whether a module requires data or sends data with events is application specific.

5 Default Events

Modules may implement automatic configuration of events (default events) which can make a module easier to configure and use. Default events would be implemented as automatic configuration of EVs based upon a module’s manufacturer's configuration, configuration of NVs or configuration of other events.

Default events behave in the same way as user configured events, it is only the way in which they come into existence which is different.

Default events must be reported by NERD and if the module supports the Teach service they must be able to be deleted or reconfigured.

6 Appendix - Discussion for Nerds

In the classic producer-consumer model, each event-number is unique and independent from other event-numbers. The CBUS developers decided that because many layout items are binary, they would use one event-number for a pair of trigger and actions, namely “On” and “Off”, and differentiate the action by having a pair of opcodes.

The CBUS Long-Events were developed when SLiM-modules were common, and the developers felt that many modules would be producers only. Therefore, it was convenient to pre-assign event numbers (ENs) to a module’s input lines (0-n), and to ensure that the event identifier was unique by including the module’s node-number as the high-part of that number (i.e.NN:EN).   Long-Events were promoted as a one-to-many solution, since one producer would connect to many consumers.

It then became clear that a many-to-many solution was required. This was mainly due to the development of throttles, since many throttles might control many accessories and locos. A solution was to invent and implement Short-Events, where the producer-NN is ignored, and only the EN, renamed to device number (DN), would be matched to other Short-Events stored in a consumer module.

These solutions work ok, and since there are 65k Short-Events, there is an ample supply for a layout.

In fact, the Long-Events are really 32-bit arbitrary numbers (albeit with the restriction that the top two bytes cannot be zero, if Short-Events are used). They can be assigned as the user likes, since modules are capable of sending any of them, and therefore they,too, can be used for many-to-many relationships.

The summary of the appendix is that advanced users may choose to push the bounds of the VLCB as they see fit. The rule “It’s your layout” still rules supreme.

7 Glossary

EN	Event Number (0-65536, 0x0000-0xFFFF).
Event	Represented as NN:EN and when transmitted onto the VLCB bus indicates a change of state.
EV	Event Variable. Used to define the behaviour of a module associated with an event.
Event Identifier	A unique identifier for events on the network. Short events use a 16 bit event number (EN). Long events use a 32 bit identifier consisting of node number and event number.
Event Index	A temporary identifier used as a shorthand for NN:EN.
Event Number	A 16 bit value specifying the purpose of an event.
Mode	Indicates what set of messages a node will respond to and how it behaves to those messages.
NN	Node Number (1-65536, 0x0001-0xFFFF)
Teach	The process of associating an Event and its EVs with a module.